A CMOS Analog Hopfield Net with Local Adaption and Storage of Weights

Aanen Abusland

February 10, 1994

Preface

The work for this thesis was conducted at the Institute of Informatics, University of Oslo. Part of the research was funded through the NIADE project.

I thank my advisor Tor Sverre Lande for always being encouraging and always offering constructive criticism of my ideas, no matter how farfetched they might seem. I have also been allowed great liberty in choice of methods and solutions, which have suited me very well.

I also thank my parents for their support, both monetary and not the least morally. Without them this thesis never would have been possible.

Contents

Part I

Background

Chapter 1

Introduction

1.1 Rationale for the thesis

The later years have seen a large interest in artificial neural networks (ANN). Mostly this is due to their computation paradigm, massive parallel computation performed by many very simple processors. The label ANN reflects that this is the computation method used by the neural systems of all living beings (those that have neural systems, that is). This is also true of brains, including the human brain.

Just as the brain of most humans have greater computational capacity than the brain of an insect, the computational capacity of a large ANN usually exceeds the computational capacity of a small ANN. Hence, the potential of ANNs will not be realized until large systems can be built.

One way to make an ANN, is to implement it on a microchip with CMOS transistors. The implementation may be either digital or analog. Electronic analog systems have greater computational density than their digital counterparts (e.g. [49]). Analog CMOS systems also have lower power consumption, especially analog circuits with transistors operating in the subthreshold region [41].

Interconnections inside an ANN, often termed weights, may either be fixed or adaptable. CMOS ANNs with fixed weights may perform very well at a specific task [38]. However, for an ANN to have some versatility, weights must be adaptable. Adaptable weights may either be computed locally on-chip, or down-loaded from off-chip. Down-loading of analog weights by shift registers is difficult due to offsets. Down-loading with addressing may be done [19], but may be fault-prone as system size increase. Also, it is easier to achieve modularity in a system when all computations are done locally. Therefore, both modification and storage of weights should take place locally.

There are several methods for computing change in weights [47]. Some require a supervisor, some do not. They also vary in complexity. The Hebbian rule [17] (also called the outer product rule), a supervised method, is the simplest.

Storing an analog weight requires an analog memory. In CMOS technology, making an analog memory amounts to storing a charge on a capacitor. The leakage from this capacitor determines the decay of the stored value. A net with short-term memory need a memory refreshing scheme, as well as re-programming in the event of power supply failure. A non-volatile memory is preferable. Trade-offs between long-term memory techniques include number of programming cycles, speed, special chip processes, special signal conditioning. In this thesis the UV-light programmable floating-gate technique is used. UV-programming may be performed with a standard CMOS process, for an unlimited number of programming cycles.

A large integrated system will experience defects, and should be fault tolerant. A net with inherent fault tolerance is the Hopfield net [22, 23]. A Hopfield net is also versatile, depending on the modification of weights [24, 56, 57].

1.2 Terms and nomenclature

Generally speaking there are two types of artificial neural networks, feedback nets and feed forward nets. A series connection of several layers is most common with feed forward nets. Connections between neurons are termed synapses, or conductances. The magnitude or value of a conductance determines the strength of the connection between the output of a neuron and the input to another. The value of this connection and the connection itself is in the litterature often termed a weight, and is represented by the notation w_{ij}, or the weight from neuron j to neuron i. The collection of weights in a net may be represented by a matrix, e.g

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}$$

The transferfunction of a weight is assumed to be linear. A weight may generally assume all real values. The strength or value of a weight may be modified according to a certain "learning" rule.

There are two general types of artificial neurons. One is the sigma neuron, or summing neuron. All inputs to the sigma neuron are aggregated, or summed, and the transferfunction of the neuron computes the neuron's output from the sum. The sum is generally called the activation of the neuron, and is denoted a_i for neuron i. Another common denotion is net_i. In this thesis the former notation will be used. Mathematically, the input to a sigma neuron may be expressed

$$a_i = \sum_j w_{ij} o_j + I_i \tag{1.1}$$

where I_i is external input to neuron i. The external input may be omitted, as we shall see. The other type of neuron is the sigma-pi neuron which also perform a multiplication to form a_i. In this thesis all neurons are of the sigma type.

A neuron, be it natural or artificial, is an amplifier. The output from neuron i is

$$o_i = f(a_i)$$

where $f(a_i)$ denotes the transferfunction of the neuron. All neurons, as amplifiers, posess gain, which may be included explicit or implicit in the transferfunction. One transferfunction type is the binary function where either (-1, +1) or (0,1) are the only possible outputs (i.e. infinite gain). The other transferfunction type is continuous, with a sigmoid or "squashing" function. Any real world implementation of a neuron has to have neurons with continuous outputs. The exact mathematical expression of the transferfunction is generally not important as long as the function is non-linear. It follows from the principle of superposition that a general feedforward net utilizing linear neurons cannot not solve other problems than a single layer feedforward net.

1.3 Chips designed

When making a test implementation of a Hopfield net we decided to keep things as simple as absolutely possible. The associative memory, described in the next section, is the basic form of the Hopfield net, and also a very interesting application. For the reason of simplicity we to use the Hebb rule to modify the weights. All chips were made in a standard double metal double poly 2μ CMOS process by Orbit.

Analog **d**evice: **H**opfield net **o**n a **c**hip (AdHoc): A chip that was made during the initial work for this thesis, containing a fully differential Hopfield net, as well as some test structures for the building blocks. It was designed spring '91.

Ultra **Vi**olet light **Te**st **C**hip (UViTeC): A chip containing UV-programmable circuits. It was designed summer '92.

Hopfield **A**ssociative **M**emory **I**ntegrated **C**ircuit (HAMIC): A chip containing a new Hopfield net, test versions of the building blocks for the net, as well as a few more UV-programmable circuits. It was designed spring '93.

Chapter 2

Hopfield nets

2.1 Introduction

A Hopfield net is a one layer fully interconnected feedback net containing two or more neurons. A neuron's output is connected to the inputs of all the other neurons, via weights. Figure 2.1 shows the principal architecture. A fully interconnected feedback net has been noted to have a parallell in a physical phenomenon known as the Ising spin model [36]. The state space of such a feedback net is N-dimensional, for N neurons. In addition there is an energy dimension, which is a holdover from the original physics model. The energy provides a method of describing the behaviour of the net inside its state space. Using the mathematics developed for the Ising model, Hopfield showed that under certain conditions, such a feedback net would have one or several stable output states, and that it would always end up in one of its stable states regardless of the starting state [22, 23]. The net may be wieved as an associative memory, where a memorized item corresponds to a stable state. The associative memory evoked considerable interest in the field of neural nets, and several papers have been published on the behaviour and storage capacity of such nets [2, 3, 11, 12, 33, 40, 45, 54, 60]. Hopfield also used a simple learning rule, the Hebbian, to modify the weights.

In later papers Hopfield, often together with Tank, demonstrated other applications for the net. Among other things, it was used to solve the Travelling Salesman Problem (TSP) [24], which is an NP-complete problem [13]. Although Hopfield's results seem somewhat difficult to reproduce directly [65], this application has given rise to some interest [1, 3, 58, 66]. The net may also be used to solve other NP-complete problems (e.g. [18, 55, 59]). This would be a very interesting target for an implementation on a chip. The disadvantage is that solving the TSP for M cities requires a net with M^2 neurons. For test implementation purposes the size of the net quickly becomes prohibitive.

Another application for the net is as an A/D converter [57], which has later been improved

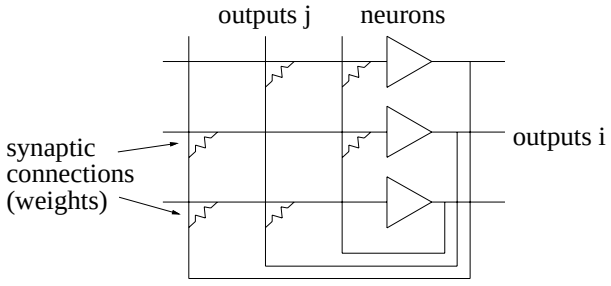


Figure 2.1: *Three neuron Hopfield net.*

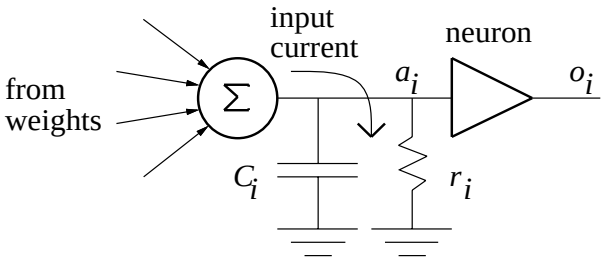


Figure 2.2: *Simple model of a sigma neuron.*

[35], as well as implemented in CMOS technology [30].

Tank and Hopfield also used the net as a Winner Take All net for speech recognition [56].

2.2 Hopfield associative memory

2.2.1 Generally

When a pattern is applied to the input of an associative memory, the memory outputs the stored pattern that most closely resembles the applied pattern. This may also be viewed as error correction; a pattern containing errors is applied and the net outputs the correct pattern. When the patterns are binary, errors or difference between patterns may be measured in Hamming distance. In other words, an associative memory outputs the stored pattern that has the shortest Hamming distance to the applied pattern. Hopfield nets are of the binary net type, even when the neurons have continous (analog) outputs. The Hamming distance a Hopfield net may tolerate between input pattern and stored pattern, depends mostly on the size of the net, the number of stored patterns, and the relation between stored patterns.

2.2.2 The energy equation

From figure 2.2 it is easy to set up a simple mathematical equation describing the activation of a neuron

$$C_i \frac{da_i}{dt} = \sum_j w_{ij} o_j - \frac{a_i}{r_i} \tag{2.1}$$

where C_i is the parasitic input capacitance and r_i is the input resistance of the neuron. If the transferfunction $o_i = f(a_i)$ acting on this input is of a sigmoid type, the Hopfield net may be described by the differential equation [23]

$$E = -\frac{1}{2} \sum_i \sum_j w_{ij} o_i o_j + \sum_i \frac{1}{R_i} \int_0^{o_i} f^{-1}(o) do \tag{2.2}$$

where $f^{-1}(o)$ is the inverse of the transferfunction of a neuron. R_i is the combined input resistance of the neuron and weights, $\frac{1}{R_i} = \sum_j \frac{1}{R_{ij}} + \frac{1}{r_i}$. The equation describes a "landscape" in the N-dimensional statespace of the net. The state of the net follows the contours of this landscape. By differentiation of equation (2.2) with respect to time, and combining with equation (2.1), Hopfield showed that the change in energy with time is always less than or equal to zero, which means that the net will seek out the "valley" closest to the starting state in the energy landscape and stay there when the local minimum is reached. Since the energy can not increase, the net is trapped in a stable state.

The effect of gain

By introducing λ as an explicit symbol for gain, equation (2.2) may be written [23]

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij}\, o_i o_j + \frac{1}{\lambda}\sum_i \frac{1}{R_i}\int_0^{o_i} f^{-1}(o)do \tag{2.3}$$

If the gain increases to infinity, the last term equals zero, and the equation reduces to the energy equation Hopfield developed for binary neurons [22]. Perhaps more interesting is what happens when the gain is decreased. This is easiest to describe using an example: Assume that a neuron's input resistance r_i is so high that it may be ignored, which is true if the input is the gate of a MOS transistor. Equation (2.3) may then be written as

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij}\, o_i o_j + \frac{1}{\lambda}\sum_i \sum_j w_{ij}\int_0^{o_i} f^{-1}(o)do \tag{2.4}$$

Further assume a two neuron net with linear weights, and weight matrix

$$W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.5}$$

Then equation (2.4) is reduced to

$$E = -o_1 o_2 + \frac{1}{\lambda}\int_0^{o_1} a_1 do + \frac{1}{\lambda}\int_0^{o_2} a_2 do$$

A partial differentiation with respect to the outputs gives

$$\frac{dE}{do_1} = -o_2 + \frac{1}{\lambda}a_1$$

and

$$\frac{dE}{do_2} = -o_1 + \frac{1}{\lambda}a_2$$

The energy function has its extreme values when the activation of one neuron is equal to the output of the other. Since weights are linear, and the neurons' outputs are sigmoid, there will be three such points on a neuron's output graph, as long as the neuron has gain greater than one. Origo will be one of these points. The upper left plot of figure 2.3 illustrates this for the function

$$o_i = \frac{2}{\pi}\arctan\frac{\pi\lambda a_i}{2} \tag{2.6}$$

The points where the curves crosses the straight dashed line, which represents the activation, will be extremal values in the energy function. The curve with a gain of one crosses the diagonal only once, at origo. If the gain is one or less, the only minimum of the energy equation is in origo, and the net will converge to this state no matter what the initial state is. For gain greater than one, origo is a saddle point in the energy equation, and the net will be unstable in this state. A Hopfield net implemented with electronic components will always be brought out of the origo state by noise.

The three other plots of figure 2.3 shows how this will influence the energy function of a two neuron net. The neurons have the transferfunction of equation (2.6). The weight matrix is the same as in equation (2.5). We observe that the stable states moves inward, closer to origo, when the gain decreases. At a gain of one, the global minimum is at origo.

The implementation of a Hopfield net made during the initial work for this thesis (the AdHoc chip) had neurons with gain equal to one. It did not work, and it took a while to find out why. Although the effect of gain is implicit in Hopfield's paper [23] and may be obvious to those with more experience in the field, it is not spelled out, which may be a pitfall for the inexperienced. Hopfield only states that as gain decrease, the net's stable states will begin to converge. Incidentally, this may be used in simulated annealing [25].

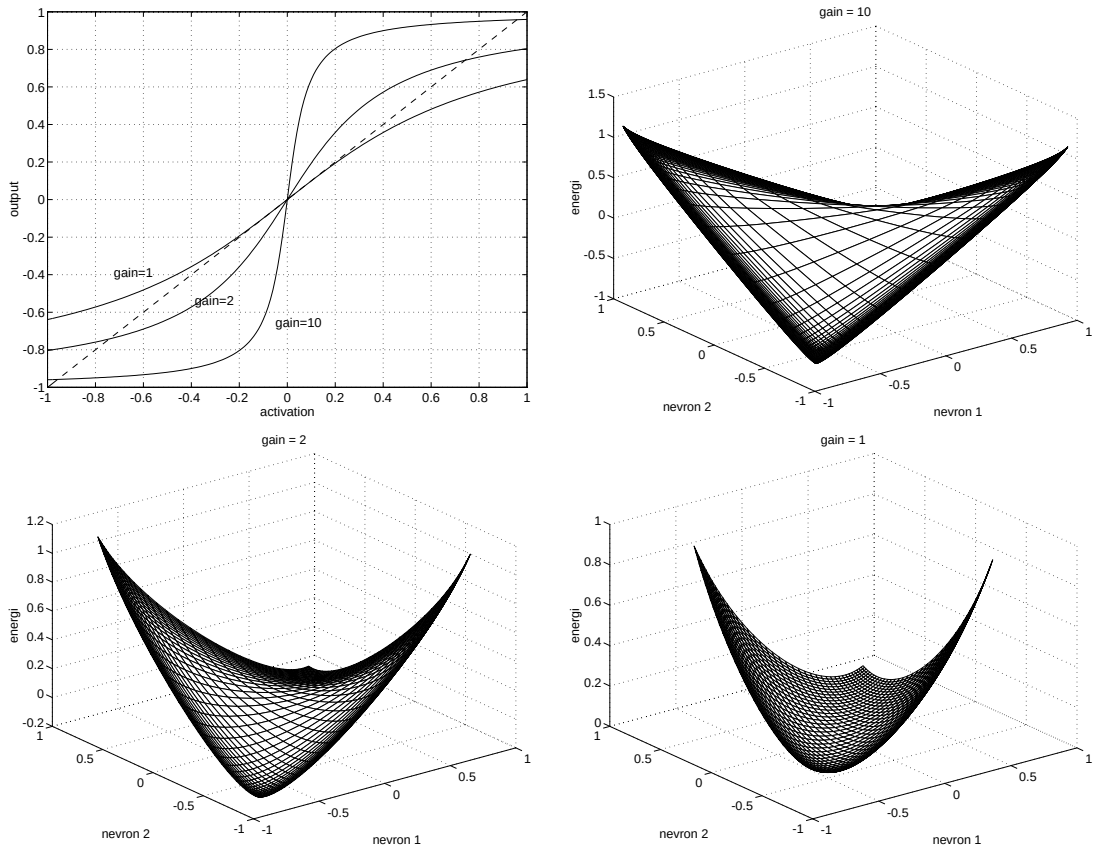


Figure 2.3: *Upper left plot shows the* arctan(.) *transferfunction with varying gain. The other three plots are maps of the corresponding energy landscape for a two neuron net.*

2.2.3 Initiating the retrieval phase

There are two ways of initiating retrieval of a stored vector. One method is used with nets that have external input units. The units may be a layer of neurons whose outputs are added to the activation in the same manner as weights. Inputs to these neurons are from the outside world. Such a layer is usually not counted, so the net is still a one layer net. The input vector is applied to these units, and the output of the net is locked to origo. The input units bias or tip the energy landscape so that when the net is released, the state of the net flows into the appropriate minimum. One problem with an implementation of this method, is that due to offsets, the net will not be locked exactly at origo, and the net will be biased.

The other method of initiating a retrieval phase is used with nets without external input units. The output of the neurons are simply locked to the input vector. The state of the net should then be on the slope of the appropriate minimum of the energy landscape. When the outputs are released, the state slide into the bottom of the minimum. This method, which do not require dedicated input units, is simpler and was chosen for implementation.

2.2.4 Capacity of the memory

People do not always agree completely upon how to define storage capacity (e.g. [46]), and published papers reflects this. However, it should be self evident that the absolute maximum capacity of the net is 2^N stored vectors, where N denotes the number of neurons. Since the Hamming distance between any stored vector and its neighbours is one, during recovery of a stored pattern, the net will answer with the applied pattern. It should also be self evident that this is not very interesting.

If the patterns to be stored are correlated, one pattern may interfere with a pattern already stored and the energy function will develop minima not corresponding to either of the original vectors. During the recovery phase, the net will sometimes end up in one of these minima, leading to an erroneous output state. In litterature, these states are sometimes termed "spurious states". Assuming that the stored patterns are vectors of length N with random sequences of bits, Hopfield found empirically that the capacity of the memory was approximately $0.15N$, where N denotes the number of neurons [22]. Storing fewer than $0.15N$ patterns keeps the number of retrieval errors under a predetermined tolerance limit. Increasing the number of stored patterns beyond $0.15N$ increases the number of erroneous retrievals, at a nonlinear rate.

Most papers seem to agree that there is an upper bound of $N/4 \log N$ on the capacity [11, 33, 60], or $N/2 \ln N$ [40] for large N. $N/0.7 \log N$ is also mentioned, for smaller N [54]. These results do not deviate very much from the capacity found by Hopfield. The results are usually subject to the conditions that vectors to be stored have binary components ± 1 with equal possibility of either state, and that they are stored using the Hebb rule.

Other learning schemes may increase the storage capacity [12, 46]. No matter what learning method is used, the capacity may have an upper bound N [2], though this is disputed [46].

The only certain way to avoid the formation of spurious states when using the Hebb rule, is to make the stored patterns orthogonal [3], which may be difficult in any useful implementation, or at least may require a lot of additional computation. For a test net however, ensuring orthogonal patterns is easy. In this case the storage capacity of the net increases.

Using other learning rules may only require that the vectors to be stored are linearly independent [3].

2.3 The size of the implemented net

We thought that storing and retrieving one pattern was going to be "easy", because the weights could always be driven into saturation during the learning phase, guaranteeing a stable state. Therefore the aim was set "much" higher; we decided to try to store and retrieve two patterns.

The maximum capacity of $0.15N$ patterns found by Hopfield suggests that to store two random patterns the net must have at least size $N = 14$, to minimize the possibility of spurious

states. Since the number of weights is a square function of the number of neurons, 14 neurons would give a large net at least for a test implementation. To decrease the number of neurons required, we decided to store orthogonal vectors, since they do not give rise to spurious states.

As we shall see, storing one vector also automatically stores its inverse. Storing two patterns therefore really means storing four vectors. How big must the net be in this case? In a two neuron network, which may store one pattern, the only possible solution is the stored vector or its inverse. This may be seen directly from the plots in figure 2.3. During the retrieval phase all applied test vectors either corresponds to a stored vector or has a Hamming distance of one to the two stored vectors. In the latter case the net will pick a solution at random, in practice influenced by whatever bias might be present in the net at the retrieval time. Therefore the smallest Hopfield net that retrieves one pattern in an interesting way is a three neuron net.

Two orthogonal patterns consisting of -1's and $+1$'s may not be stored in a 3×3 weight matrix using the Hebb rule (The Hebb-rule is explained in the next chapter). In fact, two such orthogonal patterns may not be stored in any $N \times N$ matrix where N is an odd number of neurons (This may be obvious to readers who are slightly more versed in mathematics than the author). Two orthogonal patterns will fit in a 4×4 weight matrix, but there is again the problem that all test vectors will either be a stored vector, or have the same Hamming distance to two stored vectors. Therefore the smallest Hopfield net where two patterns may be stored and retrieved in an interesting way is a six neuron net. Consequently, this was the size chosen for the implementation.

Chapter 3

The Hebb rule

3.1 Introduction

When implementing a neural net, there are many learning rules and schemes to chose from. One of the most well-known is the delta-rule, used in back propagation, one of the few learning schemes that works with nets with more than one layer of neurons. The delta-rule may also be used with Hopfield nets, as well as many other rules. However, since it was important to keep the implementation as simple as possible, the simplest learning scheme possible was chosen. This is known as the Hebb rule, or the outer product rule. Although made "famous" by D. O. Hebb in 1949 [17], the idea behind it may be traced to the previous century [26]. This was also the learning rule used by Hopfield [22, 23]. The Hebb rule is a supervised learning rule, meaning that a "teacher" has to tell the net when a pattern is learned correctly, and when it is learned well enough.

3.2 How it is used

The vector to be stored is transposed, and the rules for multiplication of matrices are used on the vector and its transpose to compute the change to the weight matrix. One way of expressing this is as [47]

$$\Delta w_{ij} = \eta o_i o_j \tag{3.1}$$

where η is a scaling factor, and o_i and o_j are the components of the vector and its transpose, respectively.

An example

Assume a four neuron Hopfield net, where we want to store a vector where the components may have the values -1 and $+1$. For instance, the vector

$$u = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

The weight matrix is formed by matrix multiplication of u with u^T.

$$W = u \times u^T = \begin{pmatrix} 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \tag{3.2}$$

If we now want to store another vector, the same method is used for the new vector, and the resulting matrix is added to the old. We may for instance want to store the vector

$$v = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

The result of applying the Hebb rule to this vector is

$$\Delta W = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix} \tag{3.3}$$

Add (3.3) to (3.2), and the new weight matrix is

$$W = \begin{pmatrix} 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \end{pmatrix}$$

Using the Hebb learning rule automatically ensures a symmetrical matrix, which as we have seen is necessary to guarantee stability. Notice that the values of the main diagonal increase proportionally to the number of taught patterns. The main diagonal represent feedback from one neuron to itself. To further ensure stability of the net, Hopfield required the main diagonal to be zero ($w_{ii} = 0$) [22]. In other applications of the net weights on the main diagonal may have other values.

The same matrix will be formed by using the Hebb rule on the inverse of the vectors used in the example,which means that for every vector the memory is taught the inverse of the taught vector is also stored. This thesis will mostly use the convention that one stored pattern equals two stored vectors, the vector applied during learning, and its inverse. During the retrieval phase, convergence to either of these vectors will be counted as a successful operation.

3.3 Weight matrix element

A Hopfield net is an extremely regular circuit. As we have seen, the weights between neurons constitutes a matrix, where all the matrix elements are identical. The main diagonal may be an exception, in which case the diagonal elements are empty (zero weight). Thus, constructing a Hopfield net largely amounts to constructing the matrix element. Equations 1.1 and 3.1 determines the computation blocks needed, and the signal flow between them. Figure 3.1 shows how the weight matrix element will look. The memory programming circuit and synaptic connection circuit will be multipliers.

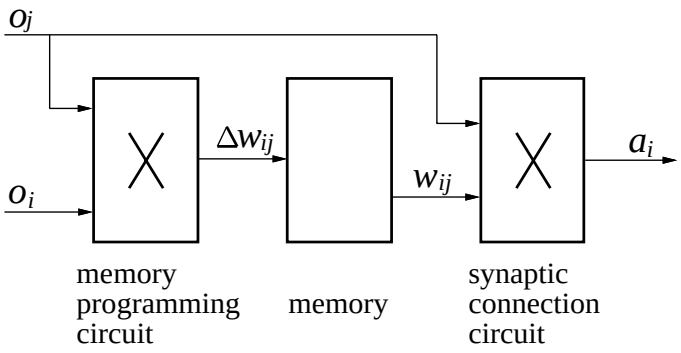


Figure 3.1: *Weight matrix cell.*

Part II

CMOS building blocks

Chapter 4

Introduction

4.1 About this part of the thesis

This part of the thesis presents the building blocks for the Hopfield net implementation. Only four subcircuits are needed: A circuit performing the synaptic connection, an analog memory, a memory programming circuit, and an artificial neuron circuit. They are presented in this order.

When reading this part of the thesis, it should become clear that the synaptic connection circuit was chosen simply because it was the only alternative seen, while far from perfect for the application. It is the only circuit directly lifted from litterature. Consequently, little work was put into this circuit, only enough to minimize some undesireable properties. The rest of the circuits are chosen, or rather built, to fit around the synaptic connection.

During the work on the net, it became more and more clear that the main problem area of the implementation was the learning process and weight storage. After designing the first chip, an abysmal failure, I realized that what existed to work with in this area did not suit the purpose very well. Therefore much of the work for this thesis has been put into the analog memory part. The memory circuit used is based upon the socalled "UV-programmable voltage reference" circuit. Chapter 6 therefore starts with a thorough discussion of this reference circuit.

The amount of space given to each subcircuit largely reflects the amount of work invested in each of them.

4.2 About simulations

All simulations in this part of the thesis were done using the HSPICE h9007 simulator from Meta-Software. The reason why is given in appendix B, along with some other comments about simulations. Listings used for the simulations are also presented.

Chapter 5

Synaptic connection

5.1 Background

What was needed for a synaptic connection, was a subthreshold multiplier with single ended voltage inputs, and as wide a linear range as possible. The output should preferably be a current for easy adding. Also, since the Hopfield net consists mostly of synapses, the circuit should be as small as possible. Many papers on multipliers have been published, some of which are expressively inteded for use in neural net implementations [8, 42, 44], and one [29] is already used to implement the synaptic connections in a Hopfield net [30]. However, the wishes for the synaptic connection circuit were difficult to fulfill all at the same time.

Some multipliers use clocking (switched capacitors)[20, 42], some are too large [21, 50, 52], some require a dual power supply [8, 48], and some have voltage outputs [9, 20, 21, 42, 61]. In addition to this, most have in common that they do not work very well in the subthreshold region, since none of them are designed for this [29, 31, 32, 44]. A few multipliers [9, 41, 52] are all based on the same Gilbert "cell" [14], and they will probably all work equally well in subthreshold. Of these, the multiplier presented by Mead [41, pp. 90–94] is the smallest. Its disadvantages are that it has a small linear range, and also requires a differential signal representation, but this may be accepted. The schematic is shown in figure 5.1. The operation of this circuit is described in detail by Mead, and here will only be discussed what is important for this application.

5.2 Gilbert multiplier

5.2.1 Circuit description, measured results and discussion

The output of the multiplier is a current, which may be described by the equation

$$I_{out} = I_1 - I_2 = I_b \tanh \frac{\kappa(V_1 - V_2)}{2} \tanh \frac{\kappa(V_3 - V_4)}{2}$$

where κ express body-effect. In figures 5.2 and 5.3 simulated and measured output currents as a function of the inputs are shown respectively. The similarity between simulation and measurements, even to the magnitude of the currents, is surprising, considering that no effort was made to adapt simulation parameters to reality.

The linear range of the circuit is approximately 150 mV. The small linear range means that offsets in preceeding circuitry will have a large influence, which is the largest undesireable property of this multiplier in this application. There are methods for increasing the linear range, such as source degradation of the differential pairs [62]. This adds complexity and reduces the operating range, and was not implemented.

The transfer curves of figure 5.3 do not cross at zero output current, but at approximately 2.5 nA. This is because the subtraction of current I_2 from I_1 is performed by the p-mirror in

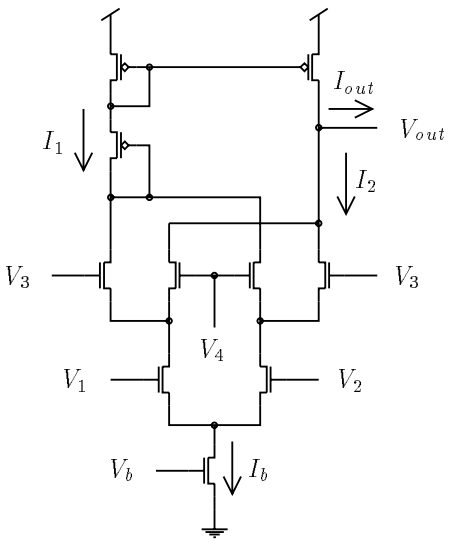


Figure 5.1: *CMOS version of Gilbert multiplier, with Benson diode.*

figure 5.1. Early-effects in the differential pairs and the output transistor due to the output voltage V_{out} results in an error in the output current, i.e. the output current depends on the output voltage. A decreasing output voltage gives an increase in output current, and vice versa. Such an effect will be systematic in a network, and will accumulate with increasin net size. In a neural net, this accumulation may bias a neuron, and lead the net to erroneous solutions.

For zero input signal, the output current due to the output voltage is zero when the errors due to Early-effects in the circuit cancel out, i.e. when the output voltage is equal to the gate/drain voltage of the diode in the p-mirror. The voltage drop over the p-diode at zero output current is approximately V_b. A DC operating point of $V_{dd} - V_b$ would be too high for this application, so the Benson diode Q_B was included to reduce the difference in drain voltage on the upper differential pairs, lowering the output voltage where errors due to Early-effects balance. Figure 5.4 shows the measured effect of changing V_{out} for constant inputs. For this particular circuit, the output current was zero for an output voltage of approximately 3.07 V and a bias voltage of 0.77 V. Due to processing variations, the balance point will vary from circuit to circuit, but the variation will be of a random nature. Random variations do not accumulate, which makes the net easier to scale.

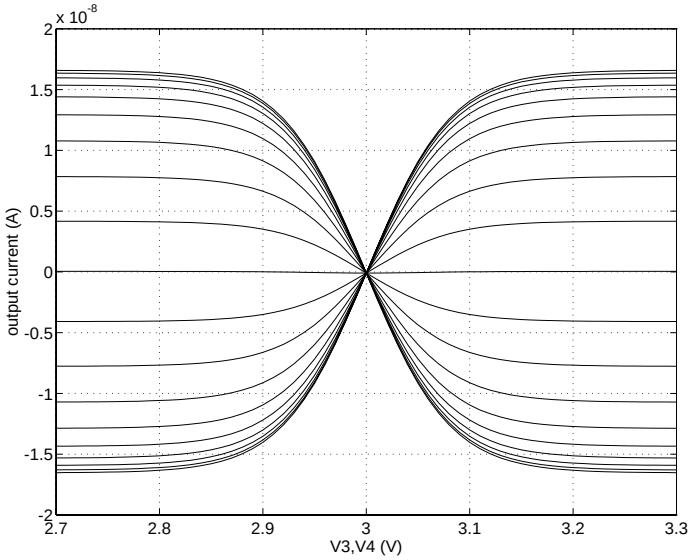


Figure 5.2: *HSPICE simulation of the Gilbert multiplier. The change in V_1, V_2 voltage between the curves is 20 mV. Output voltage is 3.05 V. Bias voltage is 0.77 V.*

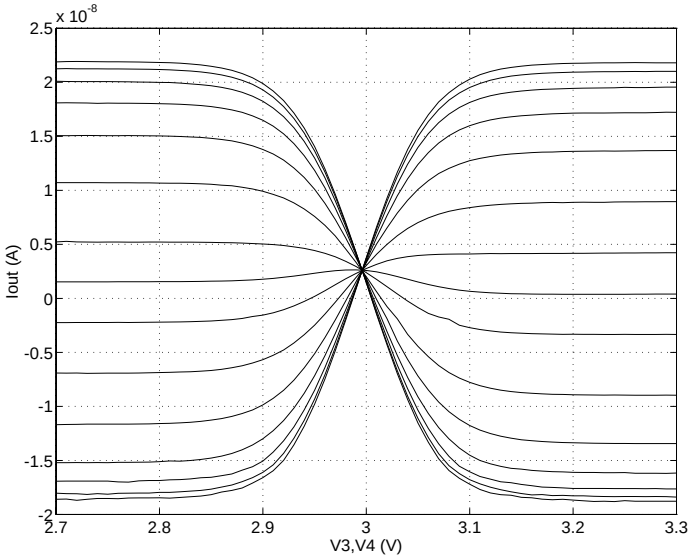


Figure 5.3: *Measured response of the Gilbert multiplier. The change in V_1, V_2 voltage between the curves is approximately 20 mV. Output voltage is 3.00 V. Bias voltage is 0.77 V.*

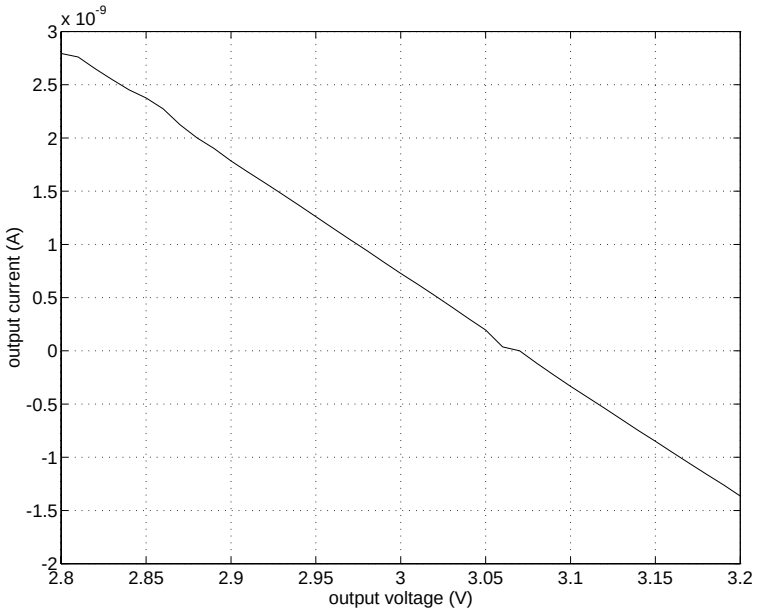


Figure 5.4: *Measured change in output current with output voltage. V_3, V_4 are at approximately 3 V, V_1, V_2 are at approximately 2 V. Bias voltage is 0.77 V.*

Chapter 6

Analog memory

6.1 Background

In analog neural network VLSI implementations, we want to compute and store analog weights locally, on the chip. To do this we need an analog memory. In a CMOS technology, this means storing a charge on a capacitor. The storage time is determined by the decay rate, the rate at which the charge leaks away. To make a long term memory the leakage must be minimal, which means that the charge must be stored on a node insulated by silicon dioxide, usually a floating gate of a transistor. During programming, the charge must cross the dioxide, while the dioxide resistance is momentarily lowered. This may be accomplished by different means, such as Fowler-Nordheim tunneling, hot carrier injection, or UV-light exposure. All techniques have advantages and disadvantages which may be considered. These will include programming speed, programming resolution, number of programming cycles, special processing requirements (non-standard or unusual chip layers), signal conditioning (high voltage) and others. For this thesis, no such considerations were done, because looking into the use of UV-light was specified for the thesis work from the very start.

6.2 UV-light mechanism

The silicon dioxide which separates the different layers of a CMOS chip from each other, is a very good insulator. But when exposed to UV-light, electrons in the valence band of silicon may surmount the energy barrier to the conduction band of silicon dioxide [64]. For this to happen, the light must have an energy greater than the band-gap (4.25 eV), requiring light with wavelengths shorter than 290 nm (UV-light), which is visualized in figure 6.1. Once out in the dioxide, the electrons respond to any electric field present. In other words, the silicon dioxide appears to conduct when exposed to UV-light, but still with a very high resistance, as we shall see later. The UV-light induced conductor is hereafter called a UV-conductor.

The UV-conductance was long assumed to be linear, but according to the recent litterature [7, 27, 37], it is quite certain that the UV-conductance is non-linear. The conductance decreases with decreasing voltages, but the exact nature of the non-linearity is yet unknown. Benson and Kerns [7] has adopted an empirical model based on the $tanh(.)$ function to fit measured data. An analytical model is presented by Maher [37].

A charge may be put on the floating-node of a transistor when an overlap between a layer with a controlled voltage and the floating-node is exposed to UV-light [10, 15, 27]. This is the same as exposing a capacitor to UV-light. The capacitor and the (potential) UV-conductor is hereafter called a UV-structure. Figure 6.2 shows a simple UV-structure, where the two poly layers constitute a capacitor. Because the upper layer will hide the lower layer, the UV-conductor will be created mainly along exposed edges. The distance between the two layers is usually smaller than the light wavelength, so the light will have no effect between the layers.

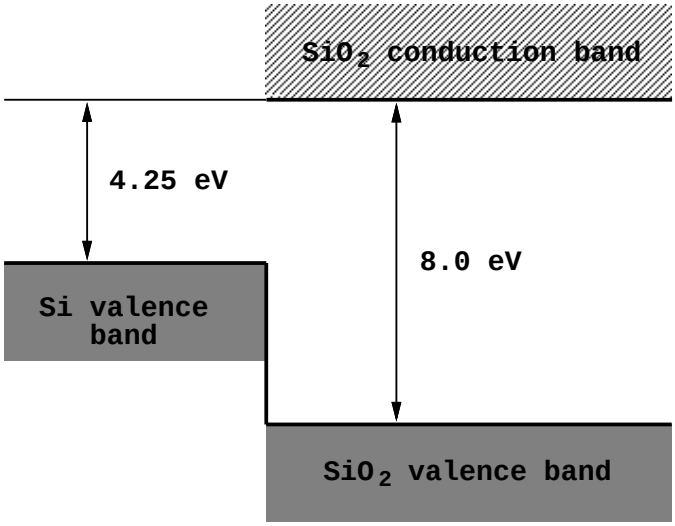


Figure 6.1: *A simple band model of the silicon/silicon dioxide interface. Electrons in silicon may be excited into the silicon dioxide conduction band by applying light with wavelenghts shorter than the band-gap. Taken from Williams [64].*

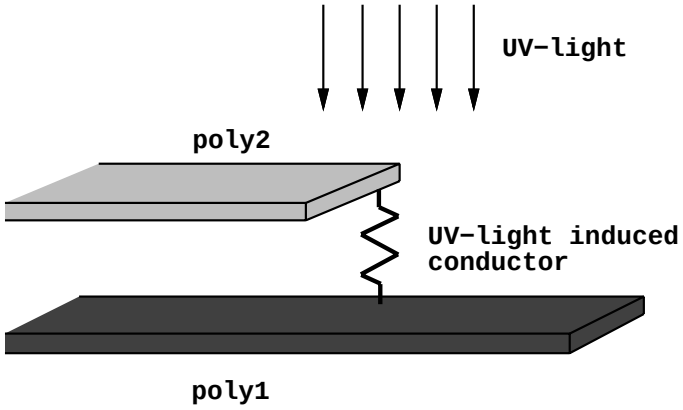


Figure 6.2: *Simple UV-structure. Electrons excited into the silicon dioxide will respond to the electric field between layers, if any. Either of the nodes may have a controlled voltage.*

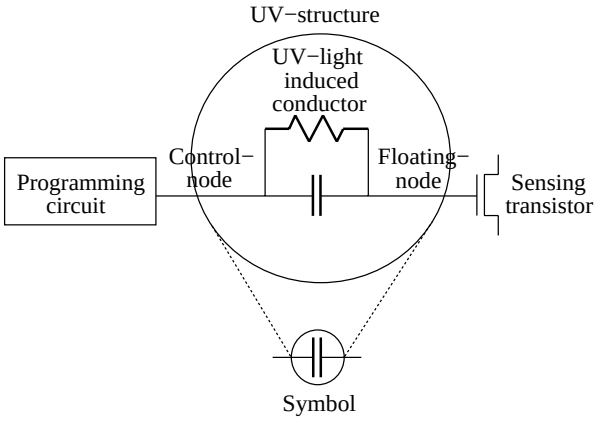


Figure 6.3: *UV-structure in the standard approach to UV-light programming.*

Circuitry which performs computations during exposure must be protected, usually by a metal layer.

When the UV-light is turned off, the dioxide regains its normal insulating properties, and the gate will retain the charge. Retention time was not investigated in this thesis because neither equipment nor time was available.

6.3 Standard approach

The way UV-programming is applied in the litterature, is to let some circuitry control the voltage of one node of a UV-structure, and to try to let the voltage of the other node, the floating-gate, approach that of the control-node. The programming circuit will do this by injecting current into the floating-gate through the UV-conductor. This is shown in figure 6.3. The programming circuit may be virtually any circuit. A single transistor [10], a NOR-gate [15], and a transconductance amplifier [27] has been used. With a slight modification, this is also the technique used by Maher for her analog memory [37].

There are two problems with this method. Firstly, the non-linear UV-conductance implies that the floating-node voltage may never match that of the control-node. Secondly, the UV-structure has a parasitic capacitor between the control-node and the floating-node. This capacitive coupling imply that when programming is done, the programming circuitry must be prevented from disturbing the floating-gate.

One method to prevent the floating-node from being disturbed, is to put the control-node at a fixed voltage. During exposure to UV-light, the floating-node voltage will swing towards the control-node voltage. This may be countered by adding another UV-structure with its control-node at a complementary voltage to the control-node voltage of the first UV-structure.

6.4 UV-light programmable voltage reference

A "floating-node" is a node without conductance to other nodes. On this node a UV-structure may store a charge, which may be sensed by a transistor. This node has a capacitance, part of which is the gate of the sensing-transistor. Another part is the UV-structure capacitance. There are also parasitic capacitors to other nodes, which may be summed into one capacitor to ground.

To create a reference voltage, two UV-structures are placed in series between the supply voltages, as shown to the left in figure 6.4. When the UV-structures are exposed to UV-light, a resistive voltage divider is created, and the floating-node is charged to an equlibrium point

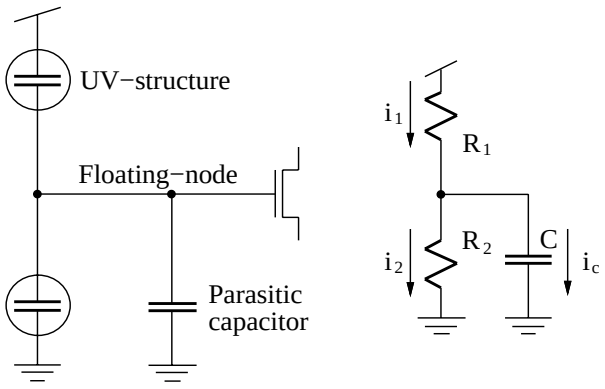


Figure 6.4: *To the left is a UV-light programmable voltage reference. To the right is the equivalent circuit during UV-light exposure.*

somewhere between the supply voltages. The value of this equlibrium point is determined by the relative sizes of the two UV-conductors.

The equivalent circuit during UV-light exposure is shown to the right in figure 6.4. From the equivalent circuit it is easy to see that the floating-node voltage may be described by the equation

$$\frac{V_{dd} - V_{fn}}{R_1} = \frac{V_{fn}}{R_2} + C\frac{dV_{fn}}{dt} \tag{6.1}$$

where V_{fn} is the floating-node voltage. Since the resistance has a non-linear dependancy on the floating-node voltage, equation (6.1) is a first order non-linear differential equation. Solving this is beyond the scope of this thesis.

Since the non-linearity is most pronounced for low voltages, the "high" voltage across the UV-conductors reduces the effect of the non-linearity. It is only the time constant that is influenced. The resulting charging curve is still very exponential-like. The detailed behaviour of the non-linearity is not very important in this application.

When the UV-light is removed, the UV-conductors disappear, and we are left with a capacitive voltage divider, where the floating-node retains the charge.

6.4.1 Design

UV-conductors are created at the edges of the chip layers. Programming time is therefore determined by the length of control-node edge exposed over the floating-node (resistance of UV-conductor), and the area of the floating-node (floating-node capacitance), and UV-light intensity. To maximize the first and minimize the second, the floating-node should be long and narrow.

Light will be reflected between layers [7, 27]. An overlap between layers is therefore required, to protect the floating-node from exposure to underlying nodes (substrate, well).

First version

The principal layout of reference circuit design 'A' is shown in figure 6.5. The floating-node was made of poly1. The exposed part of the power supply nodes was made of poly2, and placed with an overlap of 4μ over the floating-node. Exposed poly2 edge of each of the power supply nodes was 64μ long. All other circuitry was covered by metal2, and an opening, an exposure window, was placed over the reference circuit. To protect the floating-node from unbalanced exposure to metal2-edges, metal1 was placed at each end of the exposure window and connected to the power supply nodes. Exposure window edges were retracted 6μ from the power-supply edges, to avoid leakage to the floating-node. Light passing through the structure was attemped neutralized by placing the reference halfway over substrate, halfway over a well.

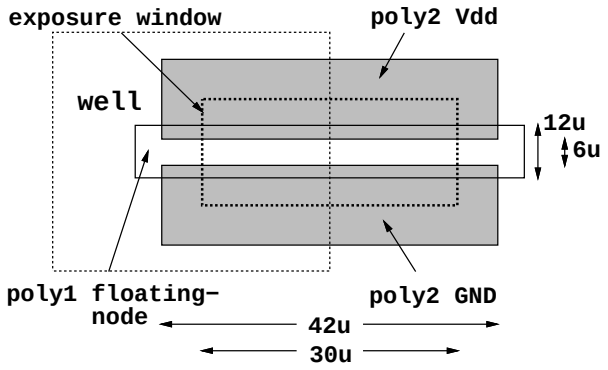


Figure 6.5: *Simplified layout of voltage reference circuit 'A'. Details such as interlayer contacts are left out for simplicity.*

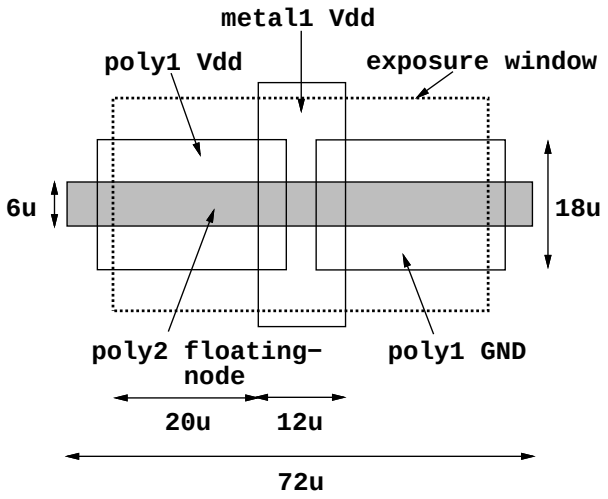


Figure 6.6: *Design 'B'. One variant of this reference was placed over substrate, and another was placed over a well.*

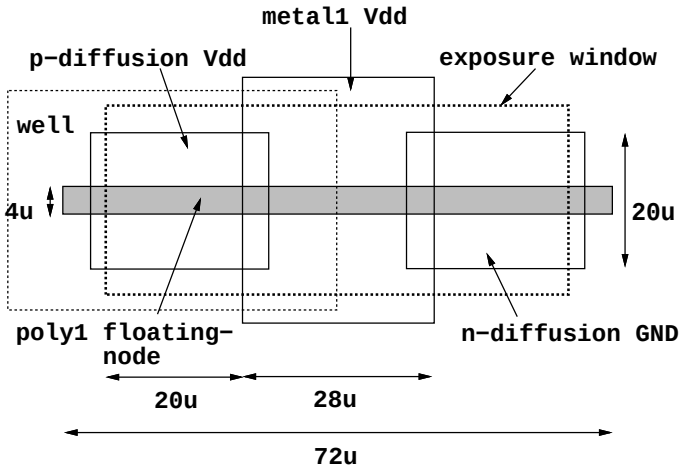


Figure 6.7: *Voltage reference layout 'C'.*

The floating-node was sensed by a wide-range operational transconductance amplifier (OTA) [41, pp. 79–81] connected as a voltage follower. The follower was connected to the floating-node by a poly1 line passing over substrate. Total floating-node capacitance, including connection to the follower, was 380 fF.

Second version

The principal layout of reference design 'B' is shown in figure 6.6. Here the floating-node was made of poly2, and the exposed part of the power supply nodes was made of poly1. This has the advantage of moving the floating-node away from the substrate, reducing both parasitic UV-conductance and parasitic capacitance to ground. Much of the parasitic capacitance will instead be to the power supply nodes, which constitute a capacitive divider. The floating-node will therefore initially be closer to its equlibrium voltage, compared to design 'A', which will give a shorter programming time. Exposed poly2 edge over each of the power supply plates was 80μ.

The 4μ gap between the two poly1 plates, where the poly2 would have been exposed to substrate, was covered by metal1 connected to V_{dd}. The metal1 had an overlap of 4μ over the poly1 plates. To give symmetric parasitics, metal1 connected to ground was placed at both ends of the exposure window. The poly1 plates were 6μ wider to each side than the poly2, to reduce parasitic conductance to ground. The exposure window was 17μ wide.

The floating-node was sensed by a simple OTA [41, pp. 70–79], connected as a voltage follower. The wire to the follower was made of metal1.

In this design, an attempt was made to reduce floating-node capacitance to a minimum, which was 76 fF.

Third version

The principal layout of reference design 'C' is shown in figure 6.6. The floating-node was made of poly1 and the exposed part of the power supply nodes was made of p-type diffusion to V_{dd}, and n-type diffusion to ground. In principle, the floating-node was the interconnected gates of two transistors, the p-transistor with both source and drain tied to V_{dd}, the n-transistor with both source and drain tied to ground. Total exposed poly1 edge over diffusion was 80μ. To avoid exposure of the floating-node to underlying nodes, the 20μ gap (minimum design) between the the two diffusion types was covered by metal1 connected to V_{dd}. As before, metal1 was placed at the end of the exposure window to provide symmetrical parasitics. The exposure window was 16μ wide.

The floating-node was sensed by an OTA of the same design as for reference 'B'. Total floating-node capacitance was 104 fF.

6.4.2 Measured results

An unpowered UViTeC chip, containing a type 'A' reference, was exposed to UV-light from a standard EPROM eraser for 30 minutes to remove initial charge from the floating-node. A supply voltage of 5 V was then applied to the chip. The EPROM eraser was placed approximately 2 cm above the chip, and the floating-node voltage was sampled at 30 second intervals. This produced the charging curve shown in figure 6.8. Defining the time constant to be the time for the floating-node to charge to 63% of the difference between initial and final voltage, the time constant of reference 'A' is approximately 120 seconds. The floating-node capacitance was used to compute the charging current, shown in figure 6.9.

A HAMIC chip, containing references of type 'B' and 'C', was run through the same procedure as described above, except that the floating-node voltage was sampled every twelfth second. Typical charging curves are shown in figures 6.10 and 6.11, respectively. The time constant of reference 'B' is approximately 100 seconds, and the time constant of reference 'C' is approximately 80 seconds.

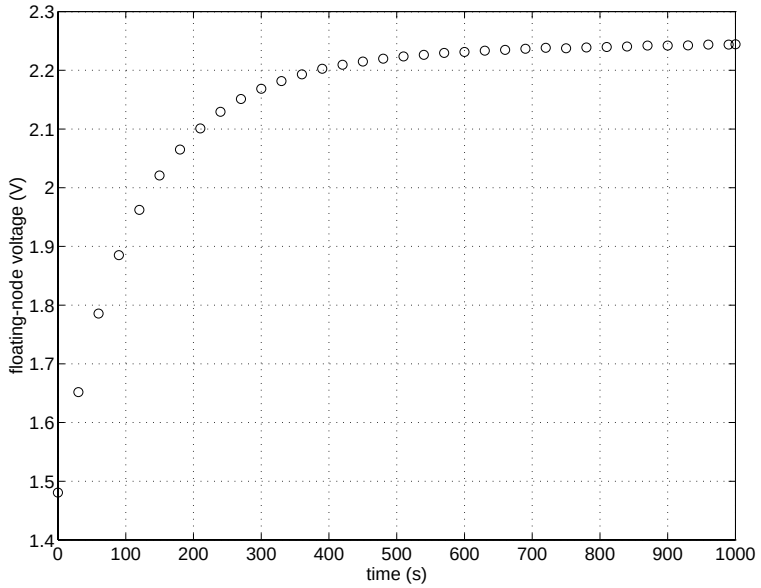


Figure 6.8: *Measured charging curve of a type 'A' voltage reference using a standard EPROM-eraser as UV-lamp. Interval between samples is 30 seconds.*

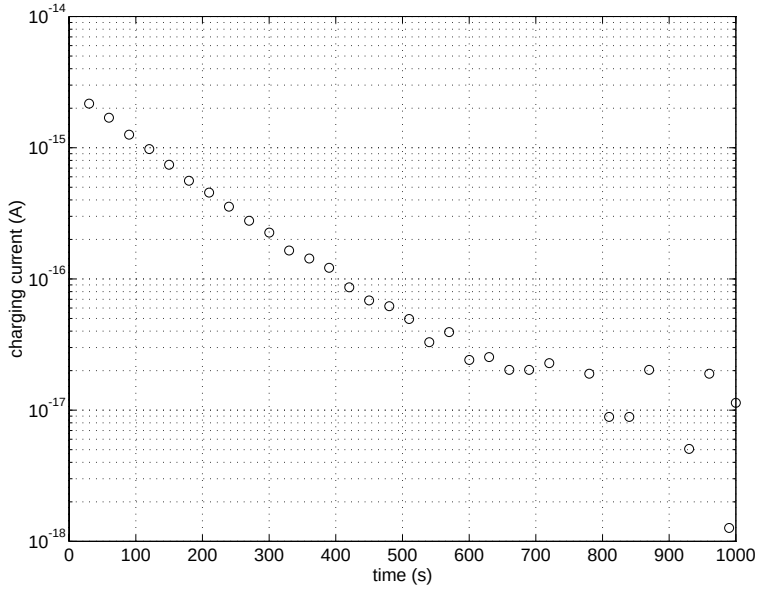


Figure 6.9: *Current into the floating-node, computed from the data in figure 6.8. The floating-node capacitance is 380 fF.*

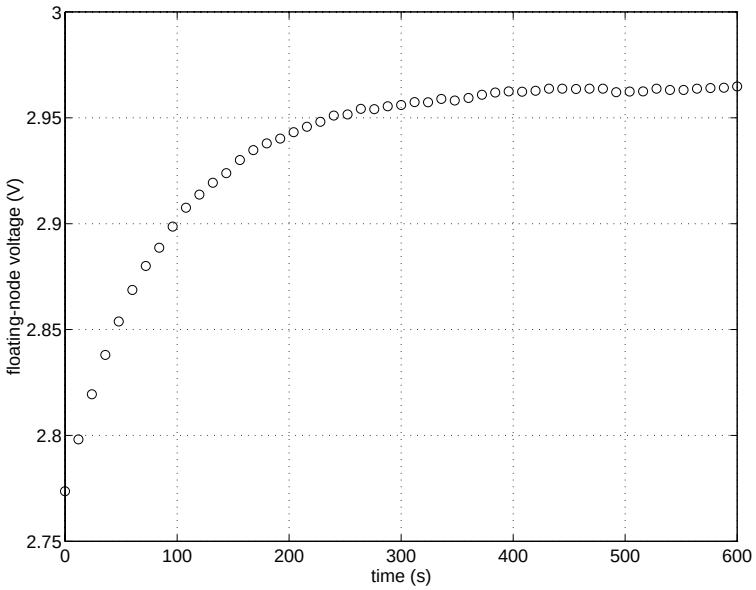


Figure 6.10: *Measured charging curve of a type 'B' voltage reference using a standard EPROM-eraser as UV-lamp. Interval between samples was 12 seconds.*

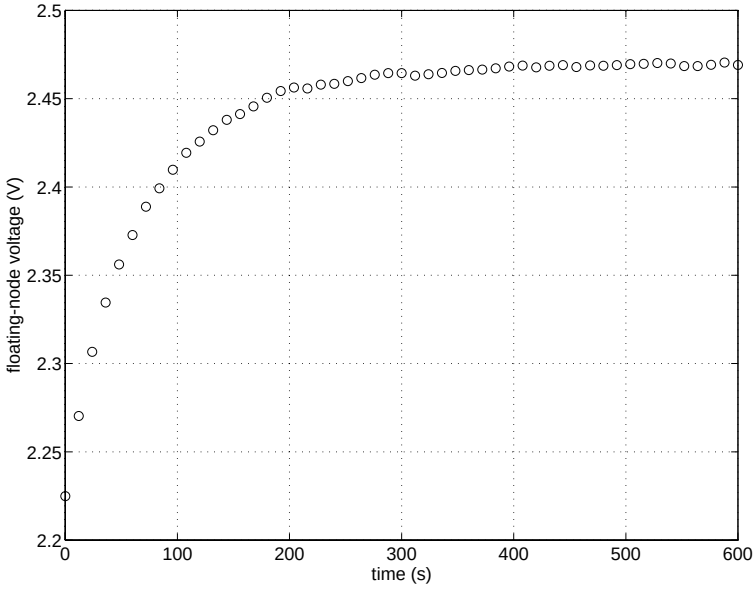


Figure 6.11: *Measured charging curve of a type 'C' voltage reference using a standard EPROM-eraser as UV-lamp. Interval between samples was 12 seconds.*

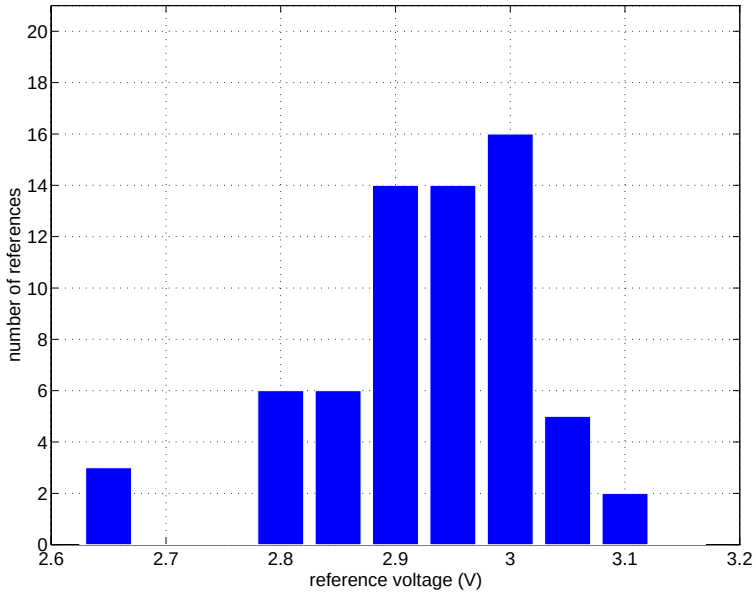


Figure 6.12: *Distribution of stored voltage on a reference of type 'B' design. Total number of references is 66, distributed on 11 chips.*

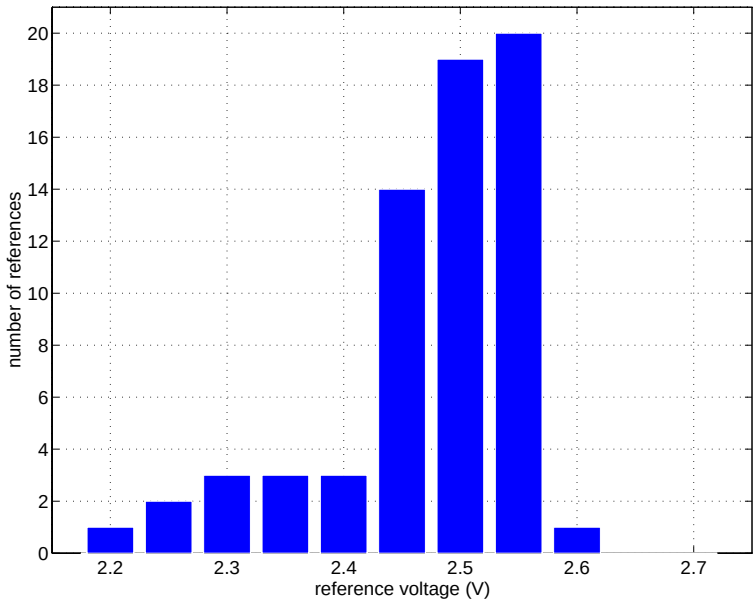


Figure 6.13: *Distribution of stored voltage on a reference of type 'C' design. Total number of references is 66, distributed on 11 chips.*

layout	mean stored voltage	median	sample deviation
A	2.266 V	2.262 V	56 mV
B	2.930 V	2.943 V	94 mV
C	2.475 V	2.504 V	89 mV

Table 6.1: *Main performance statistics for the different reference designs. The data for design 'A' was obtained from only seven circuits, on seven chips. Data for 'B' and 'C' was obtained from 66 circuits each, on 11 chips.*

Main statistical data of the three designs are listed in table 6.1. Figures 6.12 and 6.13 show histograms of the distribution of measurements of references 'B' and 'C'. Axes are scaled and bars placed such that the data are directly comparable. No histogram was made for reference 'A' because of the small body of data for this circuit.

6.4.3 Discussion

Charging curve

Because reference 'A' initially is at a comparatively low voltage, the charging current will at the start of the programming be a considerable fraction of the UV-light induced current. The charging current shown in figure 6.9 is very small, but this is in agreement with the litterature [16, 27]. It may be seen that any non-linearity of the UV-conductors does not have a large effect, the charging current appears perfectly exponential. The "noise" in the lower part of the curve shows up because the voltage change from one sample to the next, which the current is computed from, is comparable to the measuring instrument's maximum resolution.

References of type 'B' were initially much closer to their equlibrium voltage, as predicted. The charging curve in figure 6.10 starts at approximately 2.77 V, which was typical after initial UV-exposure of unpowered chips. Further UV-exposure of unpowered chips did not lower the initial value significantly. An initial value above $V_{dd}/2$ indicates that the UV-capacitor to V_{dd} is significantly larger than the UV-capacitor to ground. This may be due to mask displacement during processing of the chip. The capacitor difference is reflected in the stored voltage.

Stored voltage deviations

Table 6.1 shows that the mean stored voltage of reference 'A' is less than the nominal stored voltage $V_{dd}/2$. Clearly there is either an assymetry between the nominal UV-conductors, or the floating-node has a parasitic UV-conductance to ground. Both of these may be modelled as variations in the nominal UV-conductor to ground. Kerns [27] suspected a parasitic conductance from the floating-node. A search for correlation between length of wire to the sensing follower and deviation from the nominal stored voltage was conducted. No such correlation was found.

To look more closely at this, half of the 'B' references were placed over a well, and the other half over substrate, the hypothesis being that the potential of this underlying node would influence the stored voltage. The difference between the mean stored voltages of these two variants was 12 mV. Assuming a normal population, Student's test [6, pp. 261] was performed on the hypothesis. The hypothesis was rejected for all reasonable levels of significance. The conclusion must be that underlying nodes does not influence stored voltage in any significant way with the precautions described in the "Design" sections above.

Kerns [27] states that the UV-conductor between poly and n-diffusion should be different from the UV-conductor between poly and p-diffusion, because of different doping. From table 6.1 is seen that the mean stored voltage of reference 'B' is above the nominal stored voltage $V_{dd}/2$. This is the opposite of the result from design 'A', which was designed the opposite way. On the other hand, reference 'C' is seen to have a mean stored voltage very close to the nominal

stored voltage. More work is needed to determine whether doping has an effect. If the UV-conductor has an inherent assymmetry, it should be possible to take it into consideration when designing a reference.

Although reference 'B' and 'C' are seen to have almost the same sample deviation in table 6.1, it is clear from the histograms of figures 6.12 and 6.13 that reference 'C' generally is grouped closer around the mean than reference 'B', but that the long left "tail" of the distribution curve ruins the spread. The bins to the far left in both histograms all contain data from the same chip, making the chip somewhat suspect, although nothing wrong was seen in a microscope. Reference 'C' was expected to be a design giving less variation in the stored voltage, since the UV-structures were made of transistor gates, the best controlled part of a VLSI process.

6.4.4 General error sources

The accuracy of the reference is in theory limited by thermal noise. This is dependant on floating-node capacitor size.

$$V_N = 64\sqrt{\frac{1}{C}} \tag{6.2}$$

where V_N is the rms value of the noise in microvolts.

A more serious problem is that after programming, conditions at the other terminals of the sensing transistor will influence the floating-node voltage through parasitic capacitors. A floating-node with small total capacitance is desireable to reduce programming time, so even though the parasitics are small, they may not be negligible in a given design. It should be pointed out that this will be so for any floating-node technique, and it is not a consequence of this particular technique.

6.5 UV-programmable analog memory

6.5.1 Circuit description

When the programming of a reference circuit is done, there will be a finite charge on the floating-node. The charge may be manipulated by capacitive coupling, which may be used to make an analog memory, by connecting a coupling capacitor as shown in figure 6.14. Before programming starts, the programming circuit pulls the control-node to either V_{dd} or ground. The coupling capacitor drains charge from the rest of the floating-node. The voltage swing of the floating-node is determined by capacitive division. During UV-light exposure, the floating-node will (dis)charge towards its equlibrium point. After programming, there will be a voltage difference across the coupling capacitor, determined by the initial difference between control-node and floating-node, and the exposure time. A negative voltage on the control-node during programming, relative to the floating-node voltage, results in a programmed positive voltage, and vice versa. When the voltage across the coupling-capacitor is compressed or collapsed, the charge associated with it is distributed on the rest of the floating-node, including the gate of the sensing transistor.

The voltage across the coupling capacitor is removed or compressed by a voltage follower, hereafter termed the collapsing follower, with its reference connected to the floating-node and its output connected to the control-node. When the follower is driving, it will force the control-node to the same voltage as the floating-node. This also solves the problem of the programming circuit influencing the floating-node after programming. The programming circuit must be turned off after UV-light exposure. The offset in the follower will be scaled by capacitive division.

6.5.2 Programming the memory

An incremental programming scheme was used. The control-node was pulled either to V_{dd} or ground, and the floating-node was allowed to charge only a fraction of its dynamic range, controlled by UV-light exposure time. The programming circuit was then turned off, along with

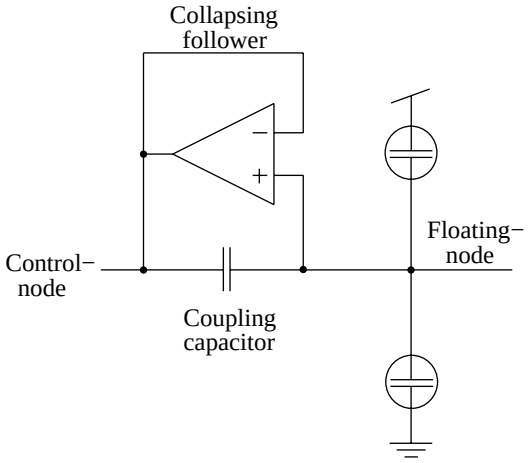


Figure 6.14: *UV-light programmable analog memory.*

1 turn off collapsing-follower bias
2 turn on programming circuit
3 apply programming pattern
4 turn on UV-light
5 wait for predetermined time interval
6 turn off UV-light
7 turn off programming circuit
8 turn on collapsing-follower bias

Figure 6.15: *Programming algorithm for incremental programming step.*

the UV-light, and the follower turned on. This required a programming time of only a fraction of the floating-node's time constant. The detailed programming algorithm is shown in figure 6.15. The programming steps may of course be put inside a loop for repeated programming.

A programming increment may be described by the equation

$$V_{fn(n)} = kV_{cn}\frac{V_{fn(n)} - V_{fn(n-1)}}{\Delta t} + V_{fn(n-1)} \tag{6.3}$$

where V_{fn} is the floating-node voltage, V_{cn} is the control-node voltage and k is the capacitive division of the control-node signal. As seen earlier, the difference term is not only exponential, but "non-linearly exponential",which makes the equation difficult, or even impossible, to solve. However, if the voltage swing of the floating-node is limited, even the exponential dependancy may be ignored, and linear programming may be assumed. In this application it was necessary to stay within a small voltage range (< 150mV) because of the limited linear range of the synaptic connection circuit. This simplification makes equation (6.3) easy to compute.

As the floating-node moves away from the equlibrium point, the non-linearity asserts itself, and programming increments gets smaller. This ensures that the memory does not diverge because of offsets during repeated up and down programming.

6.5.3 Measured results and discussion

A circuit as described in section 6.5.1 was included on the UViTeC chip. A follower with an external input was used to control the coupling capacitor during programming. The UV-structures were of the 'A' type described in section 6.4.1. The floating node was sensed by a voltage follower.

The UV-light source was a standard EPROM eraser. The UV-light was turned on all the time, so measurements were taken during exposure.

After initializing the memory, it was first programmed down, then up again, using the algorithm described above in a loop. The distance of the UV-lamp from the chip was approximately 2 cm. Exposure time between samples was 1 second. The measured curve is shown in figure 6.16. The excursion range is approximately 80 mV, which is suitable for the circuit chosen for the synaptic connection.

The floating-node of the memory was first charged to its equlibrium point with the follower turned on. As seen in section 6.4.3, the equlibrium point was not at $V_{dd}/2$, but a little lower. The control-node had a larger dynamic range towards V_{dd} than towards ground, relative to the floating-node voltage, thus the negative programming increments were larger than positive programming increments, until a new equlibrium was reached. To avoid the imbalance, the control-node should have been kept at $V_{dd}/2$ during initialization.

The UV-lamp was moved to a distance of approximately 20 cm from the chip, and the expriment repeated. The resulting measurements are shown in figure 6.17. The excursion range is approximately 8 mV, and noise is very visible on the measurements. To avoid noise problems the programming cycle was repeated several times and the results averaged. The noise is most likely 50 Hz interference from the EPROM eraser power supply, or 100 Hz intensity variation in UV-light, aliased by the low sampling rate. Also noise from other electrical devices in the environment, such as relays, may be apparent. The theoretical maximum resolution is limited by 1/f noise.

6.5.4 Differential memory representation

Why use a differential representation?

When the memory circuit was invented, it was intended for use in its single ended form. The synaptic connection circuit chosen for the Hopfield net implementation has a differential pair as an input stage. Simulations using the memory as one input and an UV-reference as the other input showed that this made the synaptic circuit's output current assymetrical, probably due

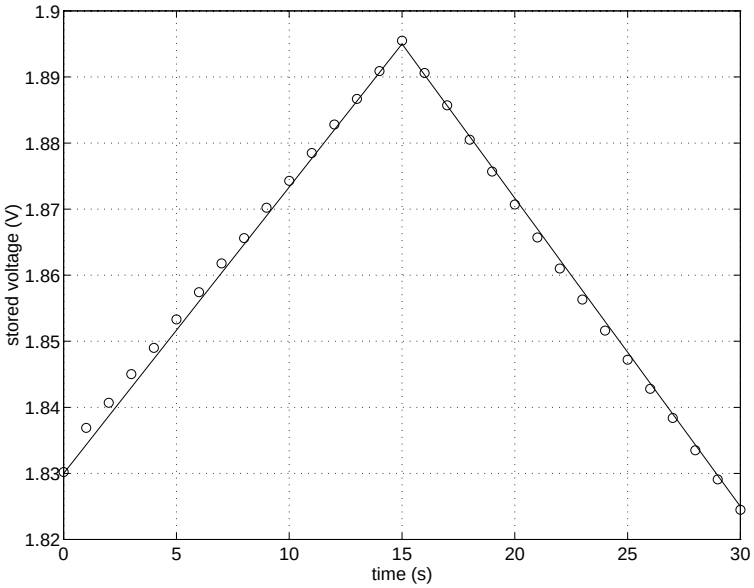


Figure 6.16: *"Medium range" programming of the analog memory. The circles are measured data. The solid line is computed from equation 6.3, with parameters computed from the measured data. The difference term was taken to be constant, but slightly different for the positive and negative slope.*

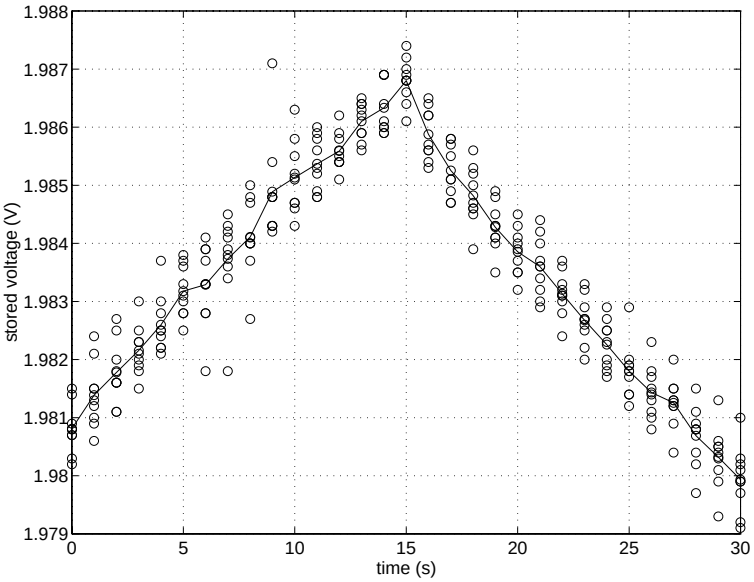


Figure 6.17: *"Small range" programming of the analog memory. The circles are measured data from nine up/down programming cycles. The solid line is the mean of the measurements.*

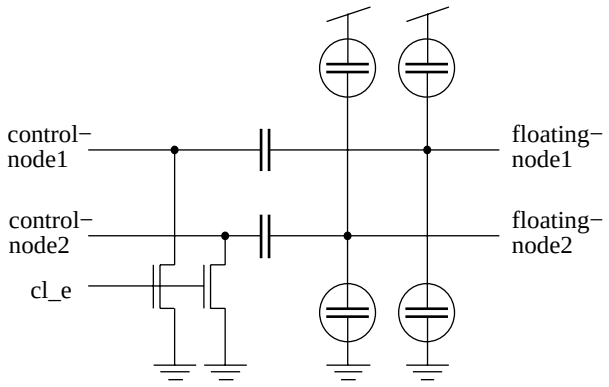


Figure 6.18: *Differential representation of the UV-light programmable analog memory.*

to Early-effect. A differential memory representation avoids this problem. Also, a differential representation reduces problems with DC level voltage and noise. On the down side is increased complexity and increased area requirements.

Circuit description

The follower in the single ended memory made the memory self-referential. In a differential representation self-reference is not necessary, as long as the two control-nodes are clamped to the same voltage when not programming. Therefore two transistors are used to clamp the control-nodes to ground, making their influence on the floating-nodes common-mode. These transistors are controlled by the signal *clamp enable* (*cl_e*). The differential memory is shown in figure 6.18.

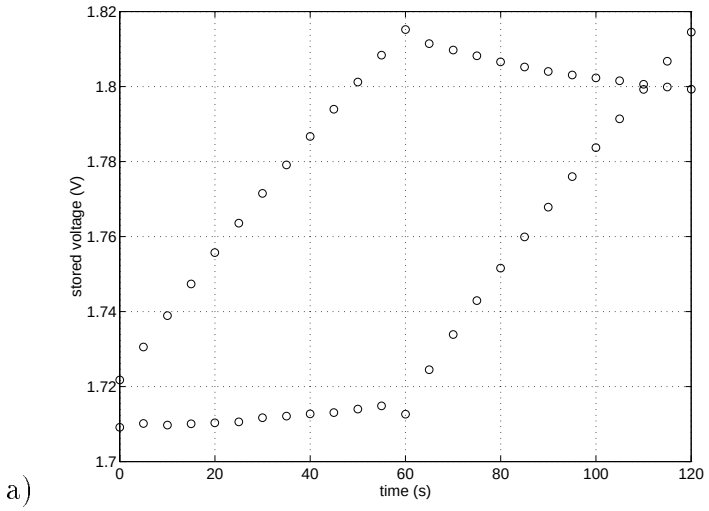
6.5.5 Measured results and discussion

A differential memory as described above was included on the HAMIC chip. The programming circuit was the one described in the next chapter. The reference circuits were of a type 'B' design. The floating-nodes were sensed by voltage followers. The EPROM eraser was used for UV-light exposure, placed approximately 2 cm above the chip under test.

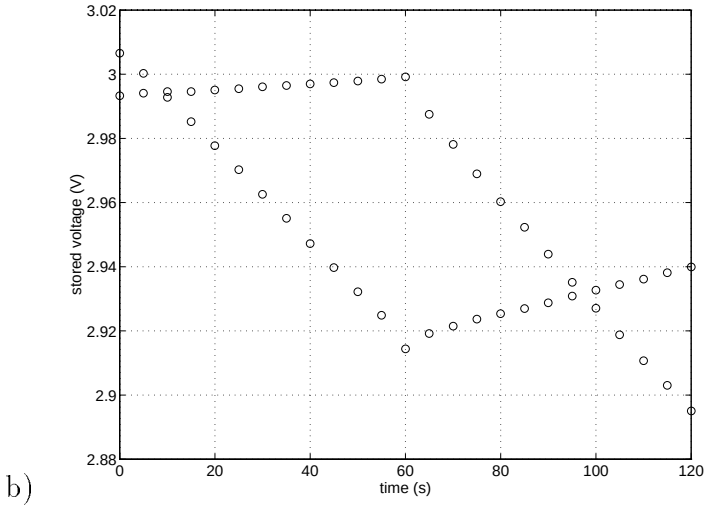
The memory was first initialized by exposing an unpowered chip to UV-light to remove initial charge from the floating-nodes. A 5 V power supply was applied, and an up/down cycle was programmed in the manner described earlier. The measured curves are shown in figure 6.19a). During the positive increment programming, the floating-node representing the positive part of the signal increases while the other floating-node voltage stays constant. During the negative increment programming, the "negative" floating-node voltage increases as much as the "positive" did in the first part of the cycle, but the "positive" floating-node voltage decreases a little again. The net result is not zero as expected from a differential representation, but a small negative value (This of course depends on which floating-node is increased during the first part of the programming cycle). Repeated up/down programming cycle will result in a "stair-step" effect, where the floating-nodes in a stepwise fashion charges towards an equlibrium value. The offset observed in figure 6.19a) is due to the lag of one floating-node behind the other. The size of the lag will depend on the number of programming increments per "stair-step".

The memory was initialized by clamping both control-nodes to ground while programming both floating-nodes to their equlibrium points. As noted before for this reference, the equlibrium point was above the nominal voltage. A measured programming cycle from this starting point is shown in figure 6.19b). We see the same "stair-step" effect as before, only the other way around, which is as expected.

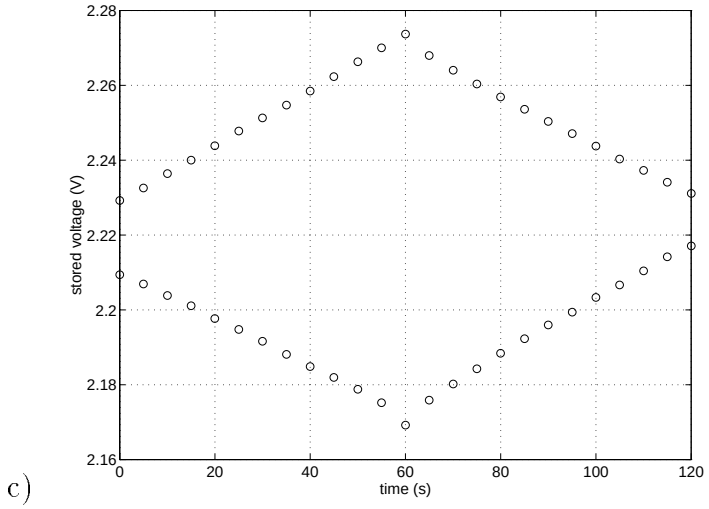
Finally, figure 6.19c) shows a memory which has been put through several programming



a)



b)



c)

Figure 6.19: *Programming the differential memory under different initial condidtions.*

cycles. Here, the "stair-step" effect is greatly reduced because the common-mode voltage is approximately at the equlibrium point. The voltage swing of the memory need to be centered around this equlibrium point for correct behaviour. Either the voltage of the control nodes must be kept at $V_{dd}/2$ during initialization, which is difficult without additional circuitry, or the memory must be put through a number of cycles before actual use. The problem of initialization will lead to problems with recovery of stored patterns in the Hopfield-net as we shall see later.

Chapter 7

Memory programming circuit

7.1 Background

In this application, the coupling capacitor of the analog memory circuit will be pulled towards either V_{dd} or ground according to a computation of the Hebb-rule, which is merely a multiplication of two signals. The magnitude of the change in the stored voltage will depend on the intensity and duration of the UV-light, as well as on the computed magnitude. In order to decrease complexity, it is logical to go one step further and require the programming circuit to provide only the sign, or direction, of the change. The magnitude of the change is then determined only by UV-light exposure time and intensity.

Since the magnitude of the inputs does not matter, we can assume binary inputs, that is, patterns of 1's and -1's. The actual voltage represented by the number will depend on the circuit chosen for the application.

In principle, any multiplier or XOR-circuit could be used for programming. A multiplier and an XOR-circuit is the same thing, but are usually designed with different objectives in mind. Most notably, a multiplier is designed to have as large a linear input range as possible, and usually a differential representation of the input. An XOR-circuit is designed to have as large gain as possible and usually has single ended inputs. We wanted the programming circuit to operate in subthreshold region, to have a rail-to-rail output swing, and high gain. It should be easy to turn on and off, and be small in area. Subthreshold operation and wide-range output is required by the thesis specification and the properties of the analog memory, respectively. One other thing required by the memory is that the output of the programming circuit must be a voltage.

A high gain programming circuit ensures that no programming pattern places the control-node in an intermediate stage, but always pulls it to the supply rails.

Turning the programming circuit off is necessary because the programming circuit must not disturb the floating-node voltage when not programming. The output of the programming circuit must have a much smaller conductance than the locking circuitry of the memory when not programming. Standard digital circuits are excluded.

It is only necessary that the circuit is able to drive a small capacitive load at a very low frequency.

As stated in chapter 5, it was difficult to find multipliers that operate in the subthreshold region, and most multiplier designs aim at increasing the linar range. The subthreshold widerange multiplier presented by Mead [41, pp. 94–96] has a suitably small linear range, but is a rather large circuit. Also, it requires a differential signal representation. Therefore, a subthreshold XOR circuit was developed, tailored specifically to the needs of the application.

7.2 Subthreshold exclusive NOR circuit

7.2.1 Circuit description

Figure 7.1 shows a subthreshold exclusive NOR (XNOR) circuit with a "wide-range" output. To get an intuitive understanding of the circuit, it may be described as follows. A XNOR function of two signals is active when both inputs are "high". This may be accomplished by two n-transistors in series, transistors Q_1 and Q_2 in figure 7.1. The XNOR function is also active when both inputs are "low". This may be accomplished by two p-transistors in series, transistors Q_3 and Q_4 in figure 7.1. These two structures in parallell will draw current in an XNOR fashion. To regulate the current, a bias transistor is put at the bottom (transistor Q_5 in figure 7.1). Transistor Q_5 will be biased in the subthreshold region, although this is not necessary for the function of the circuit.

The current is amplified by a factor of two in a p-mirror, the output of which is the top of the output stage. The mirror are transistors Q_6 and Q_7 in figure 7.1. The bottom of the output stage is a pull-down transistor (Q_8 in figure 7.1). Transistor Q_8 share gate signal with transistor Q_5. When the input transistors are active, the p-mirror pulls the output high.

The p-mirror is scaled to ensure that transistor Q_7 can pull the output high despite possible offsets. For the same reason bias transistor Q_5 is twice the width of pull-down transistor Q_8. The output current will most likely not be symmetrical, in that the circuit will be able to source more current than it can sink. Simulation results are shown in figure 7.2.

7.2.2 Problems

When input B is "high" and input A is "low", the source of transistor Q_2 will be almost at ground because of the small voltage drop over the bias transistor. Similarly when input B is "low" and input A is "high", the source of transistor Q_1 will be alsmost at ground. When the "low" input undergoes a transition to "high", the output will switch when the conductance of the input transistor approximately equals that of the bias transistor, which will occur when the input is approximately equal to the bias voltage, or more accurately, at approximately the bias voltage plus the voltage drop across the bias transistor. The same will of course apply when both inputs are "high" and one undergoes a transition to "low". When biased in subthreshold, the circuit will have a very low switching threshold. The switching threshold apparent in figure 7.2 is too low for this application.

7.2.3 Improved circuit

The switching threshold may be increased by increasing the voltage drop over the bias transistor, and thereby increasing the gate voltage required to conduct the current necessary for switching, due to body-effect. This may be accomplished by including a diode (transistor Q_9 in figure 7.3) on top of the bias transistor. Due to the body-effect, the switching threshold will be within the 1-4 V range for all useful biases. The inputs must be kept well outside this region during programming. Of course the switching threshold will still vary with bias. Neither will the "high-to-low" and the "low-to-high" thresholds be symmetrical. In this application switching threshold assymmetries does not matter, since it is only the sign of the computation (i.e. the asymptotic values) we want. Figure 7.4 shows some simulation results of this circuit.

7.2.4 Test results

The XNOR circuit described in section 7.2.3 was included on the UViTeC chip. Measurements are shown in figure 7.5. As may be seen, the circuit largely behaves as expected. In figure 7.6 the difference between some measured results and simulations is shown. The difference is surprisingly small, considering that no effort was made to match simulated transistor characteristics to reality (More about simulations can be found in appendix). Also worth noting is that there actually is

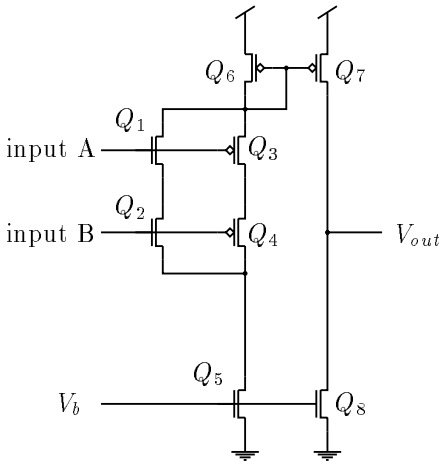


Figure 7.1: *Subthreshold XNOR-circuit.*

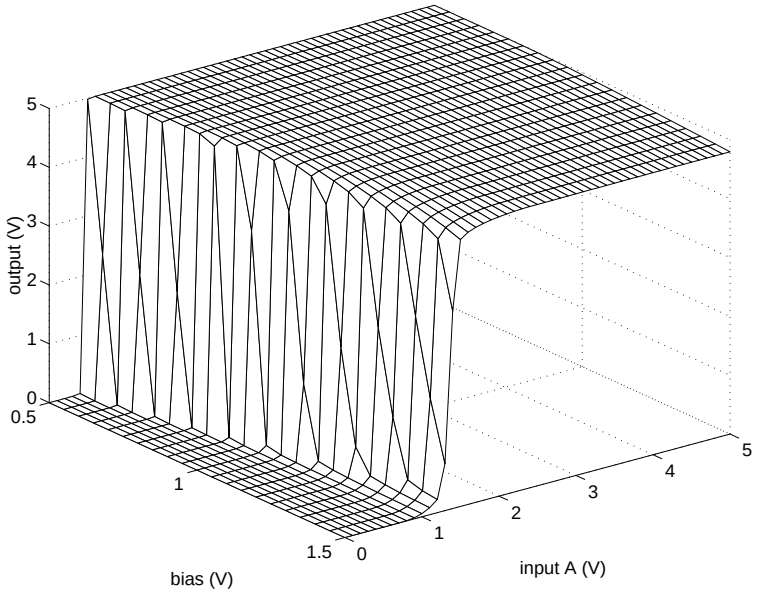


Figure 7.2: *HSPICE simulation of the XNOR-circuit shown in figure 7.1. "input B" was equal to 5 V, while "input A" was swept through the range 0–5 V at 0.1 V increment. The sweep was done for bias voltages between 0.5 V and 1.5 V at 0.05 V increments. Thus, the grid points of the surface are the simulation results.*

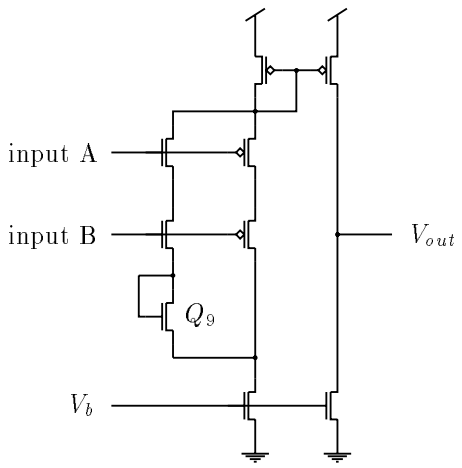


Figure 7.3: *Subthreshold XNOR-circuit with higher switching threshold.*

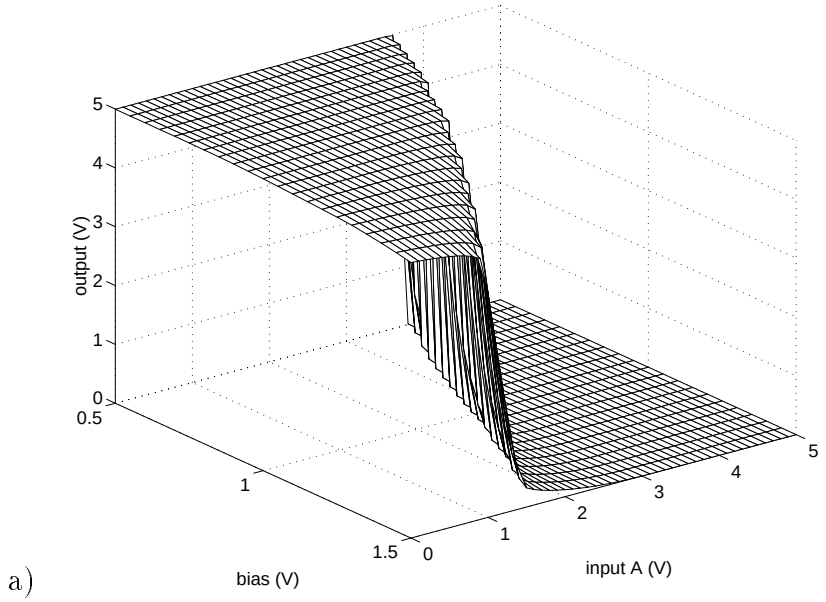
a better correspondence between measurements and simulations **below** threshold than above, which does not fit with what is commonly heard about SPICE. The largest mismatch is in switching threshold, which changes more with bias voltage in simulations than in reality.

7.2.5 Differential representation

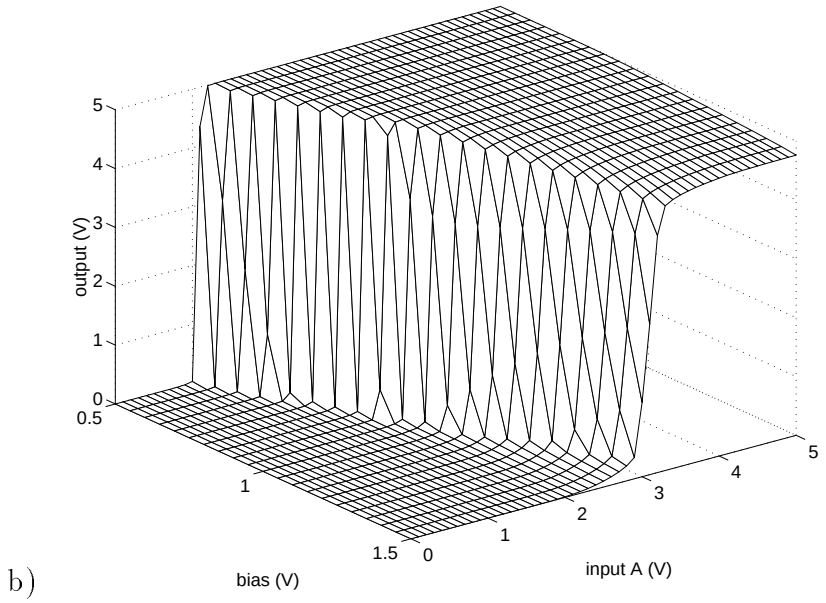
A differential memory representation requires a differential programming circuit. A differential circuit may be made by adding two mirrors and an output stage to the circuit described in section 7.2.3. The differential circuit is shown in figure 7.7. One minor difference from the earlier circuit is the placement of the body-effect providing diode Q_9. This was done solely to facilitate a little simpler layout. The function is exactly the same as before. The circuit was included on the HAMIC chip.

Measured results

The differental circuit is different from the earlier circuit only in that it has an added output stage, so the only new interesting thing to look at here is the correspondence between switching of the two outputs. In figure 7.8 a switching characteristic is shown for one particular bias. The crossing point is surprisingly close to $V_{dd}/2$. Also note that the asymptotic output values are not quite at the supply voltages, which was a property of all XNOR circuits on these chips. It was also the case for some of the XNOR circuits on the UViTeC chips. There will be a small offset in programming of the memory due to the offset between the outputs of the XNOR circuit.

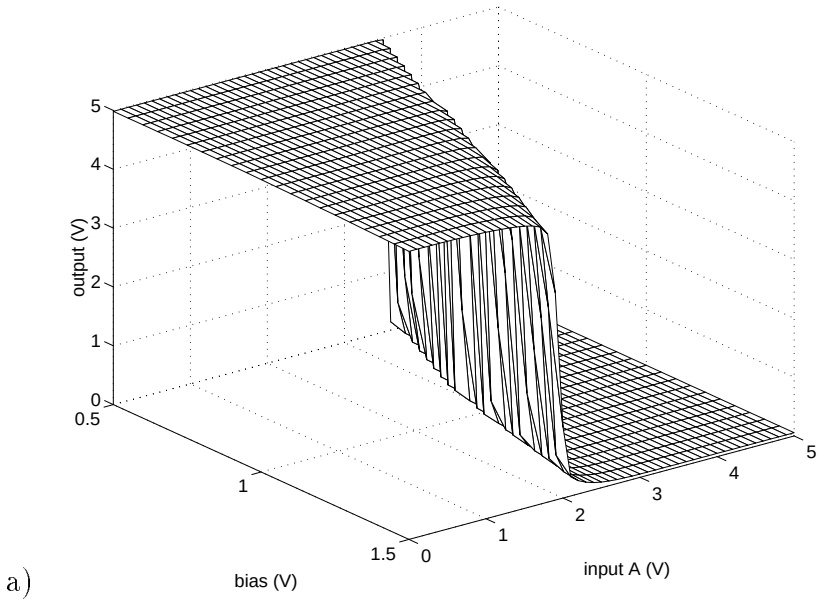


a)

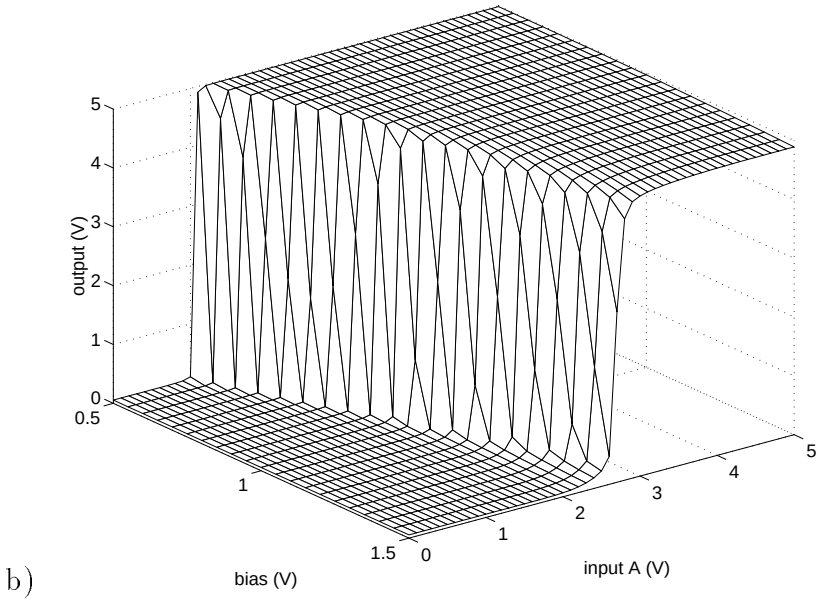


b)

Figure 7.4: *HSPICE simulations of the improved XNOR-circuit, shown in figure 7.3. "input B" was zero volts in plot a), and 5 V in plot b). The grid resolution is the same as in figure 7.2.*



a)



b)

Figure 7.5: *Measured responses of the XNOR-circuit. "Input B" was zero volts in plot a), and 5 V in plot b). The grid resolution is the same as in the simulation.*

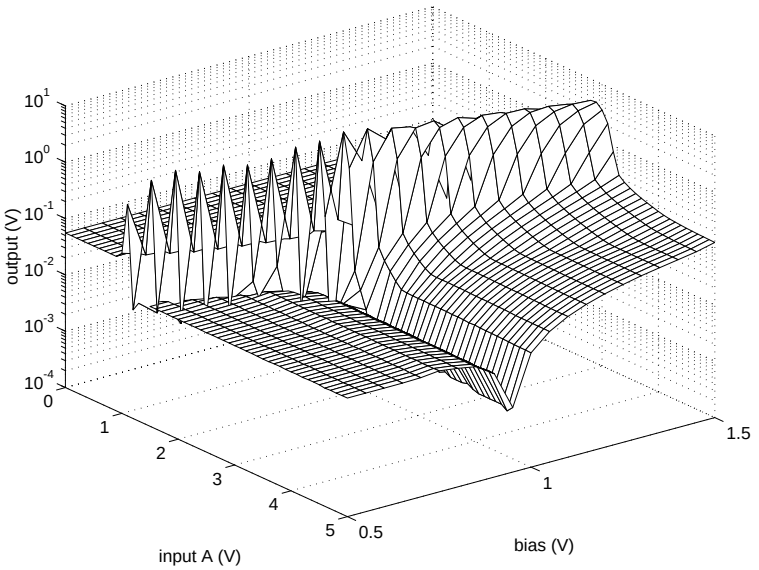


Figure 7.6: *Absolute value of the difference between the measured results in figure 7.5b) and the simulation result in figure 7.4b).*

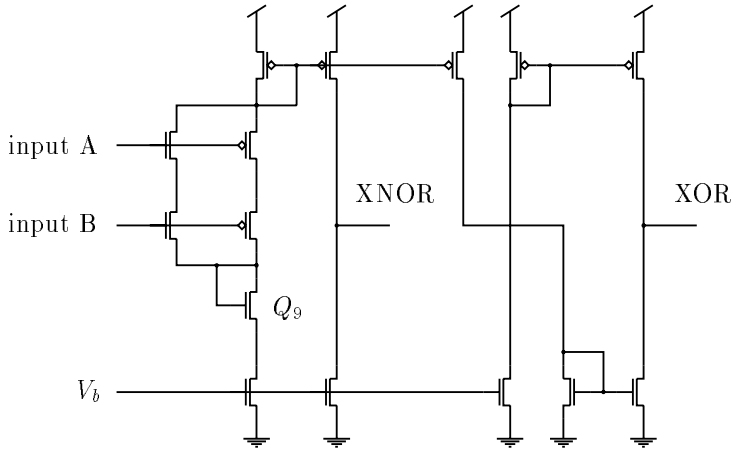
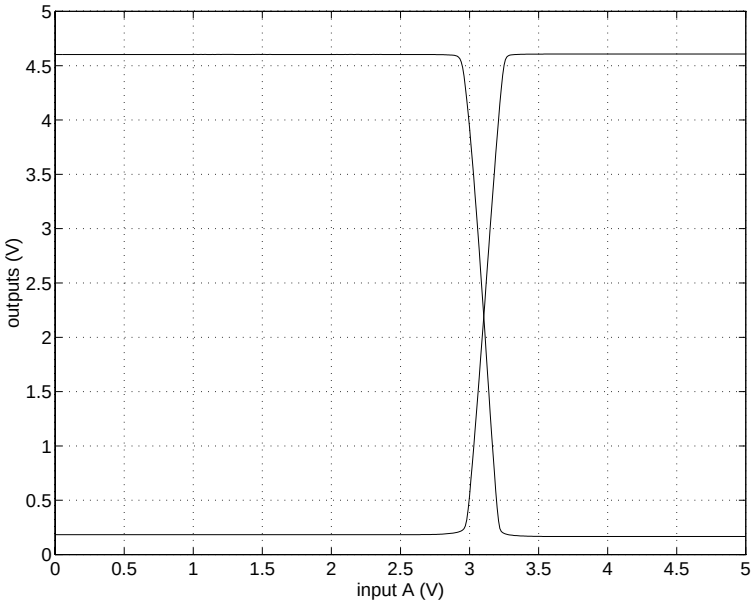


Figure 7.7: *The XNOR-circuit of figure 7.3 with an added inverting output stage.*

a)



b)

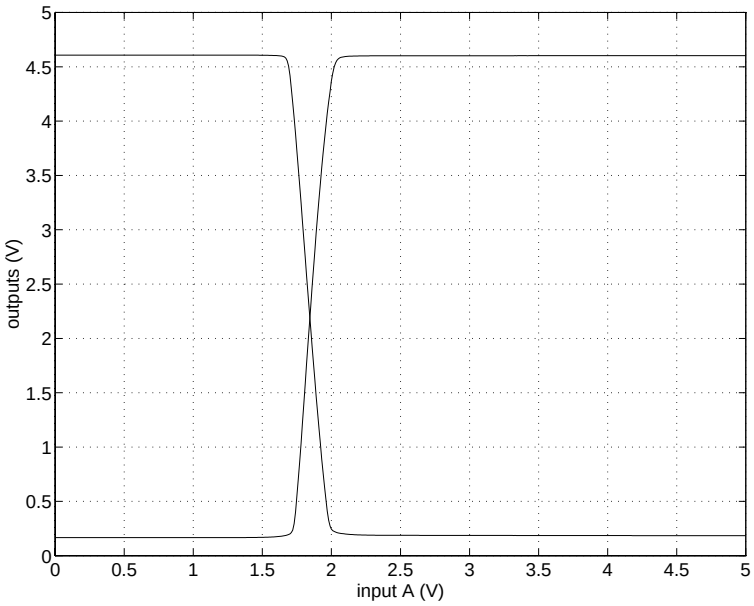


Figure 7.8: *Measured responses of the differential XNOR-circuit. "Input B" was zero volts in plot a), and five V in plot b). "Input A" was swept through the range 0–5 V. The bias voltage was 0.85 V.*

Chapter 8

Artificial neuron

8.1 Background

The output of the chosen synaptic connection circuit (the activation, or input to a neuron) is a current. The input to the synaptic connection circuit (neuron output) is a voltage. The neuron should accept a current as its input and translate this into an output voltage, in other words it should be a transimpedance amplifier. The transfer function must be monotonic, and of a sigmoid or "squashing" type. As was stated earlier, a Hopfield net, and probably most neural nets, will work with any sigmoid shaped function, so the transfer functionmay be chosen with a relatively large degree of freedom.

It must be possible to adjust the DC voltage level of the input to the neuron independent of the neuron's operation, because of the synaptic connection circuits output conductance, as described in chapter 5. A simple circuit which achieves this, is the inverting amplifier of figure 8.1 with a non-linear feedback element. The DC voltage level of the input is set with the V_{ref} input.

Regulation of the degree of non-linearity in the transfer function was desireable, but time did not allow to find out how to do this. After the last chip was sent for processing, an inverting amplifier, where the feedback element has a $sinh(.)$ function with a regulated nonlinearity, was found in the Ph.D. thesis of D. A. Kerns [28]. This circuit is intended for use as a neuron in neural nets.

8.2 Logarithmic amplifier

8.2.1 Circuit description

An inverting amplifier may be thought of as simply producing at its output the voltage necessary to source or sink the input current, through the feedback element. The feedback element may be realized by using diode connected transistors, as shown in figure 8.2. The output voltage will be

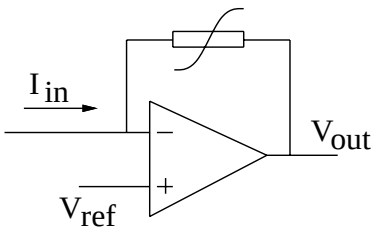


Figure 8.1: *Non-linear inverting amplifier. The non-linearity is provided by the feedback element.*

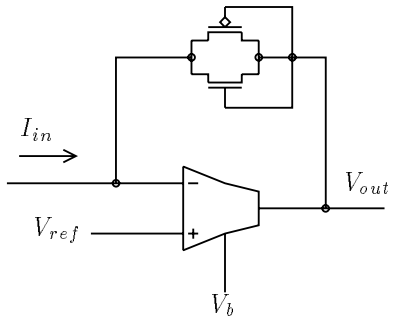


Figure 8.2: *Implementation of the non-linear feedback element. The amplifier is Mead's subthreshold widerange OTA [41, pp. 79–81], but other amplifiers should work equally well.*

a $ln(.)$ function of the input current, as long as the input current is in the range requiring only a subthreshold gate voltage. The natural logarithm is a "hard", or highly non-linear compression function, reducing a large current range to a small voltage range. The circuit will function for the entire subthreshold current range, at least six orders of magnitude, as long as the amplifier is biased to sink the input current. The drawback is that the closed loop gain cannot be set externally.

Because body-effect reduces the effectivity of the gate voltage in controlling transistor current, the output voltage swing of the circuit will be greater than the voltages normally associated with subthreshold operation of a transistor.

If the input current exceeds the threshold value, the feedback transistors enters a region where the current has a square law relation to the gate voltage [63, pp. 39]. The output will then quickly reach the power supply voltage, or as close to it as the amplifier output will go. The amplifier will not be able to sink the input current while maintaning a "fixed" input voltage. Input currents larger than subthreshold may introduce errors in the computation of a neural net.

In subthreshold operation the relation between input current and output voltage may be described by the transistor current equation presented by Mead [41, pp. 39]:

$$I_{in} = I_0 e^{\kappa V_{out}} (e^{-V_{in}} - e^{-V_{out}}) \tag{8.1}$$

where I_0 is the leakage current with zero gate voltage, κ represents body-effect, and voltages are expressed in units of kT/q. Unfortunately, the equation cannot be solved analytically for V_{out} unless the assumption is made that there is a substantial voltage ($\geq$ 100mV) across the transistors, and the input current must be split in a positive and negative part.

Assume a negative input current, and that the feedback n-transistor is in saturation. The term $e^{-V_{out}}$ then becomes small, and may be ignored. Rearranging and taking the logarithm:

$$V_{out+} = \frac{\ln(I_{in}) - \ln(I_0) + V_{in}}{\kappa} \tag{8.2}$$

The amplifier has open loop gain

$$A = \frac{V_{out}}{V_{ref} - V_{in}} \tag{8.3}$$

Solving this for V_{in} and combining with 8.2 yields

$$V_{out+} = \frac{\ln\left(\frac{I_{in}}{I_0}\right) + V_{ref}}{\kappa + \frac{1}{A}} \tag{8.4}$$

For an input current of the opposite sign, the p-transistor will be the active feedback element. The same reasoning as above holds, except that the transistor current equation is referenced to V_{dd} rather than ground [41, pp. 38]. Therefore V_{dd} must be introduced explicitly in the transfer function to get a common reference with (8.4). The output voltage then is

$$V_{out-} = \frac{-\ln\left(\frac{I_{in}}{I_0}\right) - V_{ref} + V_{dd}}{\kappa + \frac{1}{A}} \tag{8.5}$$

Generally, n-type and p-type transistors will neither have the same κ nor the same I_0. Also, κ will depend on V_{ref} because of body effect. It is not very likely that the output of the neuron will be exactly symmetrical around V_{ref}.

The amplifier may be virtually any amplifier, the only requirements are an output voltage swing close to the power supplies, and that it is able to source/sink the input current. In this application the wide-range operational transconductance amplifier (OTA) described by Mead [41, pp. 79–81] was chosen, because of its familiarity, and its ability to operate both below and above subthreshold. Due to capacitive loads, operation above threshold may be necessary, depending on speed of operation. The OTA "normally" has a gain of approximately 1000, making the fraction $\frac{1}{A}$ negligible.

8.2.2 Measured results and discussion

A circuit as described insection 8.2.1 was included on the HAMIC chip. Figure 8.3 shows the measured response of the circuit together with curves from equations (8.4) and (8.5). The theoretical curves were matched to the measurements by adjusting κ and I_0. The theory fits the data very well, except for the middle part, which is missing from the theory. The missing part is the consequence of the simplification made when the transfer function was described. The equations describing the output are not valid for small output voltages, corresponding to extremely small input currents.

The voltage swing is approximately ± 1.25 V, relative to the reference. The swing is larger upwards than downwards. Both of these are results of body-effect, and will change with the reference voltage.

In figure 8.4 the measured response is plotted together with the simulated response. The similarity between simulation and reality even to the result of body-effect is surprising, considering that no effort was made to adapt simulation parameters.

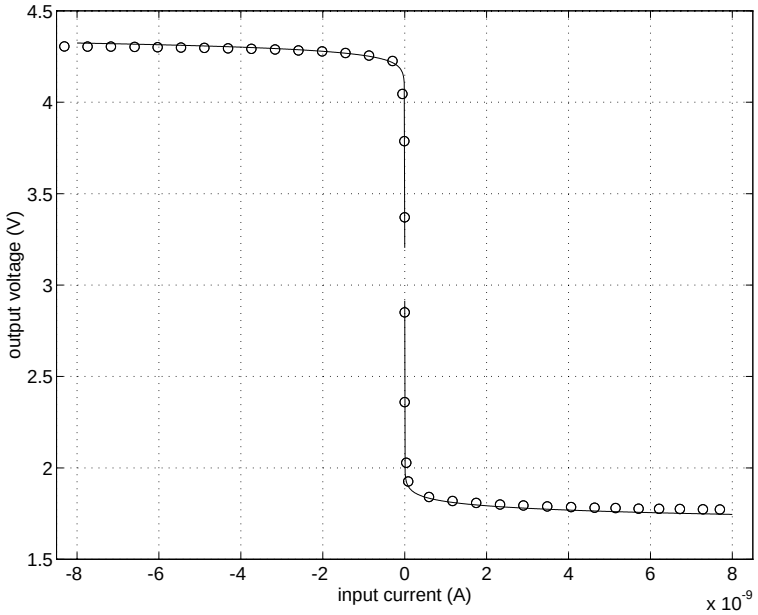


Figure 8.3: *The solid line is the theoretical response of the artificial neuron, computed from equations (8.4) and (8.5), when V_{ref} was 3 V. The circles are the measured response of the neuron, when V_{ref} was 3 V and bias was 0.85 V.*

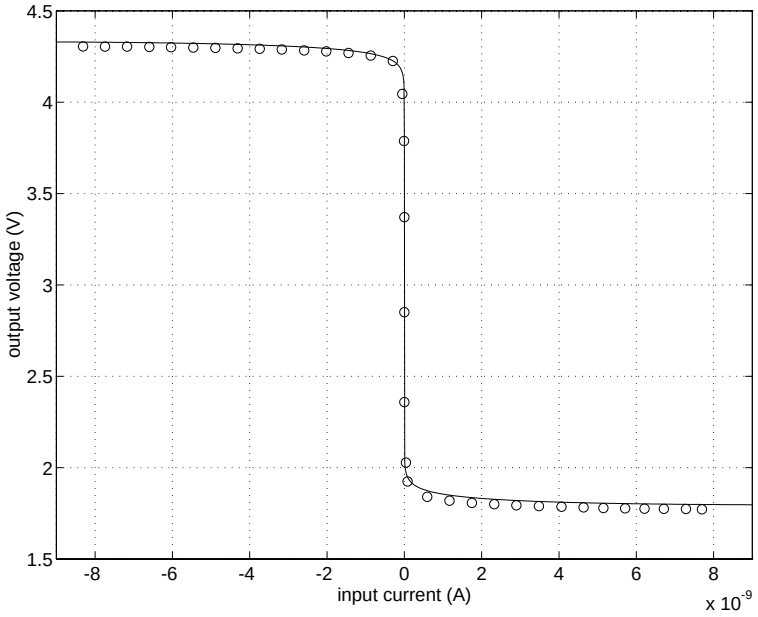


Figure 8.4: *The solid line is the HSPICE simulation result, when V_{ref} was 3 V and the bias was 0.85 V. The circles are the measured response, the same as in figure 8.3.*

Part III

The Hopfield net

Chapter 9

Implementing the net

9.1 Weight matrix element

9.1.1 Schematic

When the building blocks are ready, the blocks of figure 3.1 may be replaced by a schematic diagram at transistor level. The schematic is shown in figure 9.1. The memory programming circuit, the memory and the synaptic connection was put together in such a way that a positive $o_i o_j$ product of equation 3.1 programmed a positive increase in weight. Since the neurons were inverting, the reference voltage of the neurons (V_{ref}) was sent to the positive inputs of the synaptic multipliers, labelled *weight ref* in the figure. The output from the neuron was fed back to the inverting input of the multiplier, making the synaptic connection inverting also. A Hopfield net is symmetrical, so the inverting connection was not necessary for the operation of the net, but made thinking a little easier.

9.1.2 Layout

The available chip area was $1500\lambda \times 1500\lambda$ (total chip area was $2000\lambda \times 2000\lambda$, but 500λ in each direction was consumed by pads and pad amplifiers, in a standard tinychip padframe). Therefore little effort was made to optimize area consumption. The entire matrix element measured $81\lambda \times 183\lambda$. As may be seen, a six neuron net still left a large unused chip area.

An effort was made making the parasitic capacitance of the floating-nodes as small as possible, by keeping the wires to the multiplier short. Total floating-node capacitance was less than 100 fF. The two UV-structures were made symmetrical to get as low an offset as possible.

The entire matrix element was designed as one unit of the "butting" type. The layout of the element is shown in figure 9.2.

9.1.3 Test results and discussion

Figure 9.3 shows the output of the synaptic connection circuit when the memory was programmed up and down a few cycles. Ideally the output should be the same as the one from the test of the synaptic connection circuit (figure 5.3). We see that this is not quite so. The curves are considerably more "disorganized". The disorder may in part be due to noise on the floating-nodes. As was seen earlier, noise on the floating-nodes may easily be a few millivolts, peak-to-peak. Because of the large transconductance of the Gilbert multiplier, even a few millivolts at the inputs makes a noticeable difference in the output current.

When making a similar weight cell with floating-nodes, Benson and Kerns [7] noted an increase in current around the zero crossing point, which they did not try to explain. Something of the same effect may be seen in figure 9.3, but with the opposite sign. Change in output voltage indicates that the current is dependant on output voltage, that it is the same result of

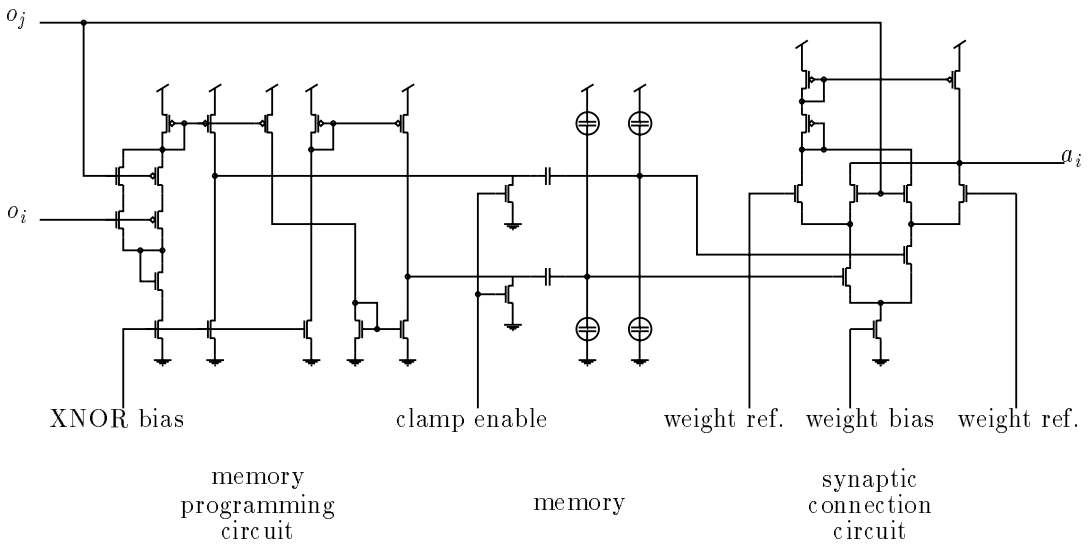


Figure 9.1: *Complete schematic of the weight matrix cell.*

Early-effect as noted in chapter 5.2, but that the increase is somehow spread over a broader range by the use of floating-gates.

9.2 Energy equation

Since the transferfunction of the neuron is known, as well as the connectivity of the net, the energy landscape of the net may be computed. The energy function of a Hopfield net is [23]

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} o_i o_j - \sum_i \frac{1}{R_i}\int_0^{o_i} f^{-1}(o)do \tag{9.1}$$

The input to the neuron is a current, so R_1 will be ignored and set to one. More serious is the fact that the function describing the behaviour of the neuron, arrived at in the previous part of the thesis, is discontinous. Continuity is a requirement for the energy function. The discontinuity will be ignored, too. When the output of a neuron, which is the input to the weights, exceeds approximately ± 100 mV, relative to V_{ref}, the output of the weights saturates. Saturation will be ignored, and it will be assumed that the synaptic connection circuits always are in their linear range, so that weights may be taken to be integers. By splitting the energy equation, as was done with the neuron output function in chapter 8, and inserting the appropriate part of the neuron function into each of the pieces, the energy landscape may be computed.

Two neuron net example

As in the example of chapter 2.2.2, the energy map of a two neuron net with the weight matrix of equation (2.5) may be computed. When the active feedback transistor is the n-type, for both neurons, equation (8.1) and (8.4) may be inserted into (9.1) to yield

$$\begin{aligned} E = \quad & -\frac{\ln\frac{I_{in1}}{I_0}}{\kappa}\frac{\ln\frac{I_{in2}}{I_0}}{\kappa} \\ & +\frac{I_0 kT}{q}\left(\frac{1}{\kappa}\left(e^{\kappa V_{o1}}+e^{\kappa V_{o2}}\right)-\frac{1}{\kappa-1}\left(e^{-V_{o1}(\kappa-1)}-e^{-V_{o2}(\kappa-1)}\right)-\frac{2}{\kappa}+\frac{2}{\kappa-1}\right) \end{aligned} \tag{9.2}$$

where all voltages are expressed in $\frac{kT}{q}$, and all voltages are referenced to V_{ref}. The equation may be simplifed, but having it on this form keeps it simple to see where the different terms come

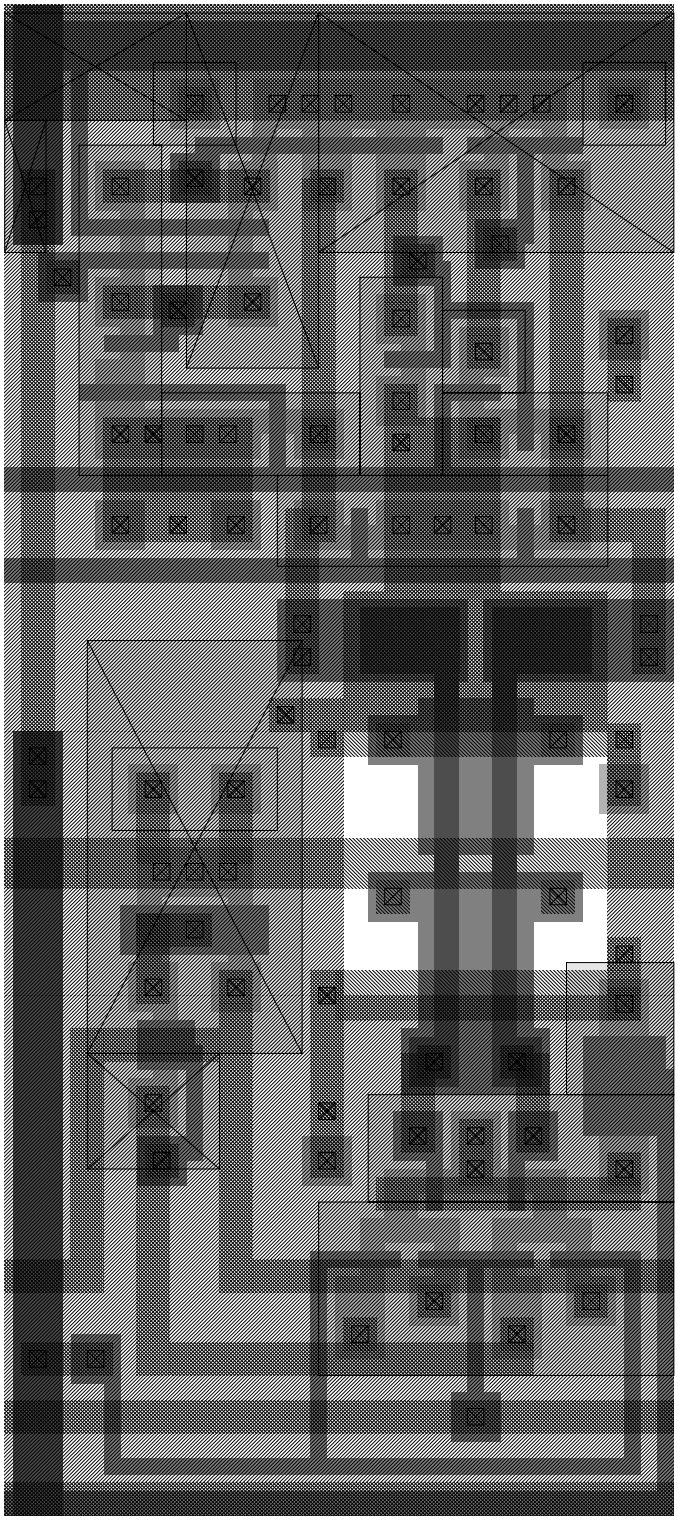


Figure 9.2: *Layout of the weight matrix cell.*

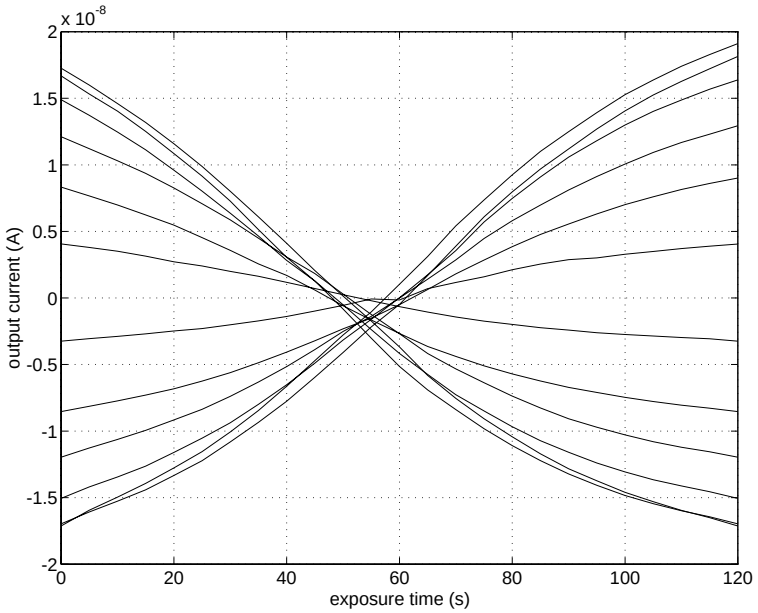


Figure 9.3: *The measured response of the weight cell when the memory was put through a few up/down cycles. The o_j input to the cell differ approximately 40mV between each curve. The x-axis shows UV-light exposure time, the y-axis shows the change in output current.*

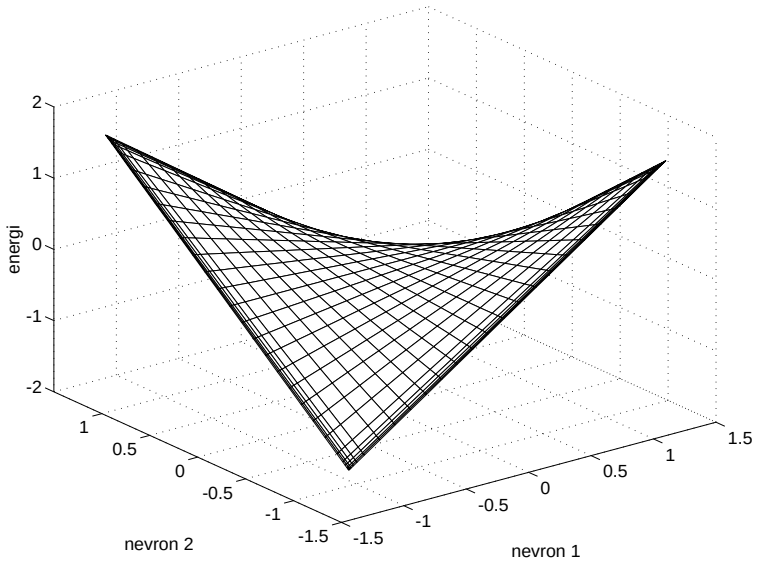


Figure 9.4: *Energy landscape of a two neuron net. The threshold for the neurons (V_{ref}) was defined to be zero.*

from. Because of the splitting of the neuron transferfunction, (9.2) is valid for only one quadrant of the landscape. For the other quadrants there are corresponding equations. Figure 9.4 shows a map of the total energy landscape.

Chapter 10

Applying patterns

10.1 Locking the neuron outputs

10.1.1 Learning phase

During the learning phase the neuron outputs must be locked to the element values of the vector to be taught. The locking may be done by pass gates, controlled by the signal *lock enable* (*L_e*) as shown in figure 10.1. When open the pass gates shorts the neuron outputs to an externally applied vector. The lock vector is applied externally from (analog) input pads. For the teaching phase, the applied vector have element values of 0 V and 5 V, or as near as possible. The programming circuit presented earlier is a digital circuit, and it is desireable to have as large a margin to its switching threshold as possible.

10.1.2 Retrieval phase

Originally it was planned to use 0 and 5 V for the elements of the test vector during the retrieval phase also, but this always gave a lot of errors during simulation. Because of the particular construction of the neuron circuit, forcing the output of a neuron also influence the voltage of its input. The influence was at first assumed not to matter, because the input to a neuron is a current. But the input line has a parasitic capacitor, which is charged during the influence of the test vector. During the retrieval phase, the parasitic capacitor must be discharged, which bias the net. The bias often leads the net to erroneous states. The "solution" was to decrease the voltages of the elements of the test vector. By simulation, it was found that letting "± 1" be represented by approximately ± 20 mV, relative to the reference voltage of the neuron, gave good results. The starting state is very close to origo in the energy landscape, where the borders of all the minima of the energy function meet.

There was some worry about how accurate the pass-gates would be for such small voltages, but measurements of the transfer curve of several gates showed that difference from gate to gate was small, even between chips.

10.2 Lock signal

As shown in figure 10.1, the pass gates for the locking inputs were controlled by inverters. Earlier experiments with hybrid analog/digital chips done at the institute has indicated that this mixing may not be straightforward [34]. The switching of digital circuituits may create noise, influencing the analog part of the chip through the power supply lines and through substarte. Therefore there was some worry that the inverters controlling the pass-gates would influence the net during the retrival phase. The inverters were given separate power supply wires in to the guard ring around the chip, which was supplied directly from the power pads. The guard ring, among other

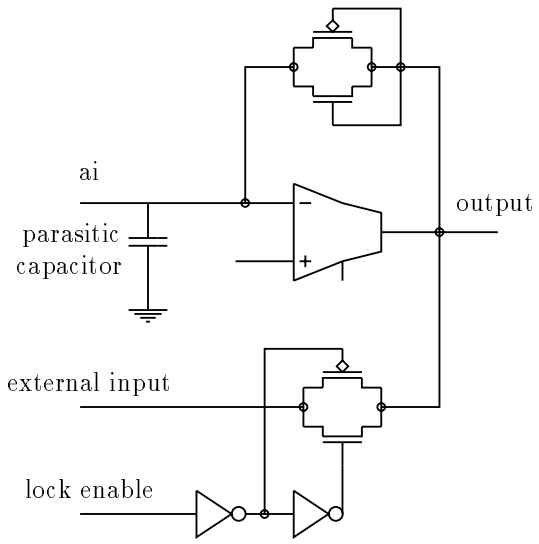


Figure 10.1: *External values applied via pass-gates.*

things, acts as a capacitor which damps switching noise from the inverters. Also, the inverters were run at quite low frequency, around one kiloHertz. Effects that are easily traced to the switching of the inverters have not been discovered.

Chapter 11

Results

11.1 Retrieval of one vector

An unpowered chip was exposed to UV-light for approximately 15 minutes, removing any initial charge from the UV-memories. Then power was applied, and *cl_e* was disabled, freeing the control nodes of the UV-memory. *l_e* was enabled, locking the neuron outputs to the external inputs. The vector $v = [1, 1, -1, -1, 1, 1]$ was applied to the input, and the chip exposed to UV-light. During exposure, one UV-memory was monitored, and when the charged difference on the floating-nodes were approximately 40 mV, the UV-light was turned off. V_{ref} was set at 3.05 V, the neuron bias at 0.85 V and synaptic connection bias at 0.75 V. *cl_e* was enabled, locking the control-nodes of the UV-memories. A 2 kHz 50% duty-cycle square wave was applied to *l_e*, locking the net to the external inputs once every 500 μsecond for a duration of 250 μsecond, initiating retrieval. Test vectors were cycled through all the 2^6 possible combinations.

Figure 11.1 shows the measured response of a neuron released from locking. Since the input to a neuron is established when it is released, the output was expected to seek out its final state at maximum (constant) slew rate. It can be seen that this is not quite the case. At approximately 3.5 V there is a "knee" in the output, which is difficult to explain.

Figure 11.2 shows the output of the net for test vector no. 64, $[1, 1, 1, 1, 1, 1]$. When the net is released, it seeks out the closest stored vector, in this case v. The net succesfully retrieved the stored vectors for all applied test vectors. The complete measurements for all applied test vectors are shown in appendix A.1. After re-initialization, other patterns were stored and succesfully retrieved.

11.2 Retrieval of two vectors

The same chip was put through the same initialization routine as described above. In addition to vector v, vector $u = [1, 1, 1, -1, -1, -1]$ orthogonal to v was also programmed. When test vectors were applied, it was found that the net always converged to one of the two stored patterns, but that in approximately 50% of the cases the net converged to u when it should have converged to v, or vice versa. The errors were thought to be due to the initialization method used on the chip. The remaining 50% of the test vectors gave succesful retrieval of the correct patterns.

As shown in the UV-memory chapter, the memory programs assymetrically if the initial value is not right. The net would then learn one vector better than the other, making one minimum of the energy function deeper than the other. The uneven learning would also shift the borders between the minima. Since initial state of a retrieval phase is very close to the border, it would on occasion be on the wrong side, and the net would end up in the wrong state.

In an attempt to initialize the UV-memories properly, the chip was repeatedly programmed with all 2^6 possible vectors. Teaching all vectors makes the weights zero, but the common-mode state of a UV-memory is different, and closer to the best initial state for program-

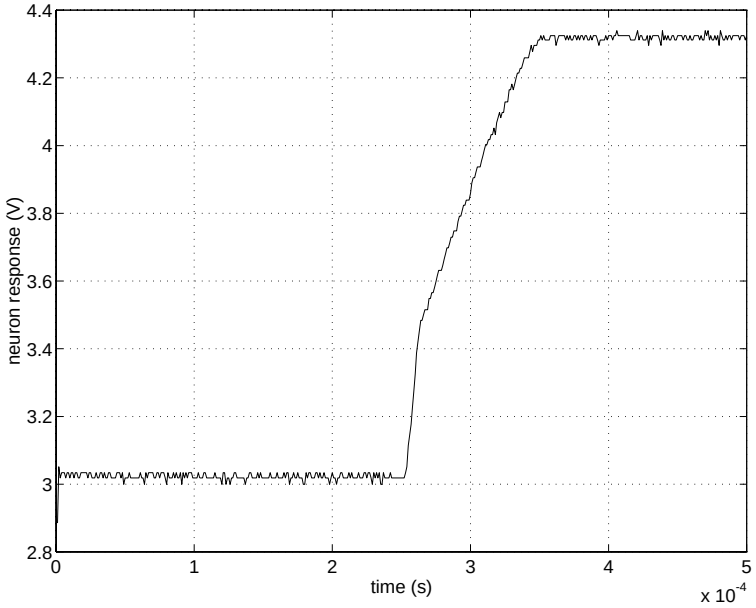


Figure 11.1: *Measured response of a neuron released at* $t = 250\mu s$.

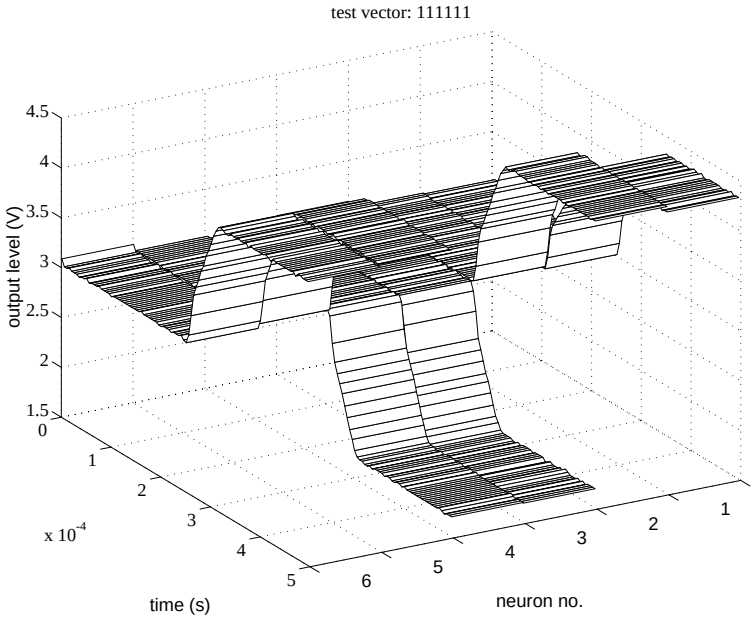


Figure 11.2: *Measured response of the net during a retrieval phase.*

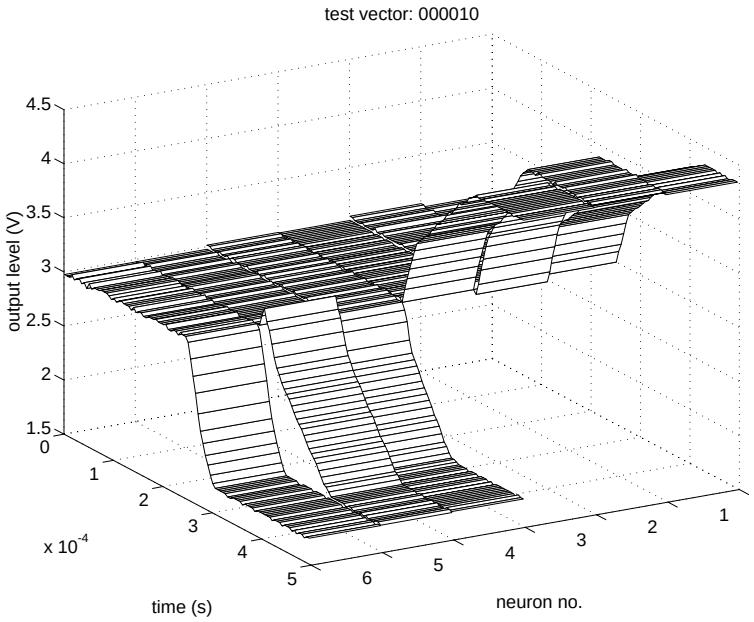


Figure 11.3: *Retrieval of the correct pattern.*

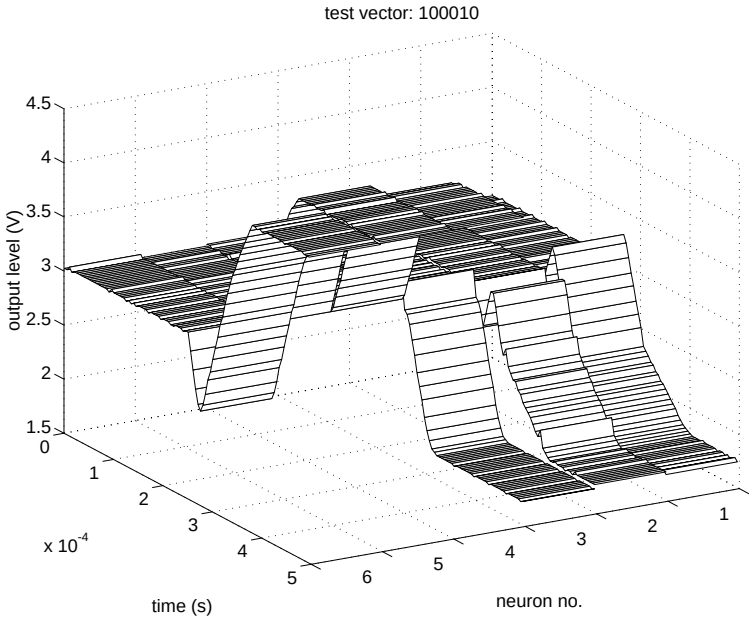


Figure 11.4: *Retrieval of the wrong pattern.*

ming, as described in the UV-memory chapter. Subsequent programming and retrieval of u and v gave an increased rate of succesful retrievals, to approximately 75%. The remaining 25% of the test vectors gave retrieval of the wrong pattern. Figure 11.3 and 11.4 shows two typical results. In figure 11.3 the net converges to the state $[-1,-1,-1,1,1,1]$, which is the closest to the starting state, and so represents the correct pattern. The starting state is $[-1,-1,-1,-1,1,-1]$, which has a Hamming distance of two to the final state. In figure 11.4 the net converges to the state $[1,1,1,-1,-1,-1]$, which has a Hamming distance of three to the starting state $[1,-1,-1,-1,1,-1]$. The closest stored vector, which should have been retrieved, was $[1,1,-1,-1,1,1]$ at a Hamming distance of two to the starting state. Appendix A.2 shows the complete series for all test vectors.

In the specific cases, the net would show a tendency to pick one vector above the other, e.g. the series in appendix A.2 shows a predominance of convergence to v. Which vector being dominant varied between initialization/learning cycles.

After once again initializing the chip by repeatedly programming all 64 vectors, the input to the chip was switched between vectors u and v at a frequency of one kiloHertz while programming. Thereby both vectors were guaranteed equal programming time. This did not increase the succesful retrieval rate significantly beyond that already achieved.

11.3 Discussion

Making one minimum of the energy function deeper than another shifts the border between those minima. This may be avoided through proper initialization of the chip. Also offsets in the UV-memories and synaptic connection circuit will shift the borders between minima. As seen in the chapter about UV-memories, offsets were somewhat larger than expected, and larger than what was used during simulation of the net. Due to the influence of the test vectors on the activation of the neurons, the net has to be started close to origo of the energy function, where the borders of all the minima of the function meet. Due to offsets, or unevenly learned patterns, some of the starting states will be on the wrong side of the border, and the net will converge to the wrong resting state. Doing the procedure of learning and retrieval many times always yilded either approximately 50% or approximately 75% succes rate, and nothing but these rates. There is probably a mathematical way of explaining this, based on the energy function.

Giving the inputs a "noise" margin would probably eliminate these errors. There almost certainly is a way to computed the required "noise" margin, based on the energy function. One way of implementing an improved noise margin would be to start the net futher from origo in the retrieval phase, which would require another implementation of the neuron, a relatively minor change. A better solution, involving a more major change in the net, would be to improve the memory, perhaps with an offset reduction scheme.

Chapter 12

Summary

12.1 Conclusion

In this thesis modular building blocks for a Hopfield associative memory have been presented. For the biases used, the power consumed by the circuits was in the nanoWatt range. In principle, it would have been possible to build a net of arbitrary size, even wafer scale. The bulding blocks were used to implement a six neuron net in a 2μ CMOS process. The net was intended for storage and retrieval of two patterns. The net sometimes has problems distinguishing between stored patterns during retrieval, but it clearly is able to store the patterns, and so may be counted at least a partial success. With some minor changes, it probably would have been completely succesful. Time did not allow such a change to be implemented.

On a larger scale the problems encountered during testing of the net may be said to be relatively superficial. A more fundamental deficiency is that the learning paradigm behind the construction is incremental, i.e. discrete. Also, inputs and outputs of the net are digital. This is also the case of other nets built at this institute [51], and elsewhere [7]. It is only the retrieval phase of the net that is truly analog in operation, which means that we are not taking full advantage of the analog computation paradigm. New types of learning methods which works in analog way are necessary, as well as net that have analog inputs/outputs. Some investigation into this direction is taking place [5], and in my opinion is the way to go.

12.2 Further work

As it stands now, the maximum programming speed of the associative memory is approximately one pattern per second, limited by the speed of the UV-memory. However, programming speed is independent of network size. The long programming time means that only at waferscale integration may the UV-programming method compete with more traditional methods of downloading. Clearly a task for the future is reducing UV-programming time.

During the work on the building blocks, some novel circuits were discovered, including the analog memory. These were based on the UV-light programmable voltage reference, which may be worthwhile in itself. One possible application is offset trimming of circuits such as amplifiers. Offset trimming is an interesting field, which also has industrial potential. It would be fun developing the UV-programming in this direction.

Bibliography

[1] Abe S:
Global Convergence and Suppression of Spurious States of the Hopfield Neural Networks.
IEEE Trans. Circuits and Systems 4, April 1993, pp. 246–257.

[2] Abu-Mostafa Y., St.Jaques J.-M:
Information Capacity of the Hopfield Model.
IEEE Trans. Information Theory 4, July 1985, pp. 461–464.

[3] Aiyer S. V. B., Niranjan M:
A Theoretical Investigation into the Performance of the Hopfield Model.
IEEE Trans. Neural Networks 2, June 1990, pp. 204–215.

[4] Anderson J., Rosenfeld E. (eds.):
Neurocomputing – Foundations of Research.
MIT Press 1988.

[5] Atiya A., Abu-Mostafa Y. S:
An Analog Feedback Associative Memory.
IEEE Trans. Neural Networks 1, January 1993, pp. 117–126.

[6] Bhattacharyya G. K., Johnson R. A:
Statistical Concepts and Methods.
Wiley 1977.

[7] Benson R. G., Kerns D. A:
UV-Activated Conductances Allow For Multiple Time Scale Learning.
IEEE Trans. Neural Networks 3, Vol. 4, May 1993, pp. 434–440.

[8] Botha T:
CMOS Analogue Current Steering Multiplier.
Electronics Letters 6, Vol. 28, 12th March 1992, pp. 525–526.

[9] Bratt A. H., King D:
Integrated Analogue Voltage Multiplier Combining MOS and Bipolar Transistors.
Electronics Letters 20, Vol. 27, 26th September 1991, pp. 1852–1854.

[10] Cauwenberghs G., Neugebauer C. F., Yariv A:
Analysis and Verification of an Analog VLSI Incremental Outer-Product Learning System.
IEEE Trans. Neural Networks 3, Vol. 3, May 1992, pp. 488–497.

[11] Dembo A:
On the Capacity of Associative Memories with Linear Threshold Functions.
IEEE Trans. Information Theory 4, July 1989, pp. 709–720.

[12] Farrel J. A., Michel A. N:
A Synthesis Procedure for Hopfield's Continuous-Time Associative Memory.
IEEE Trans. Circuits and Systems 7, Vol. 37, July 1990, pp. 877–884.

[13] Garey M. R., Johnson D. S:
Computers and Intractability: A Guide to the Theory of NP-Completeness.
W. H. Freeman & Co. 1979.

[14] Gilbert B:
A Precise Four-Quadrant Multiplier with Subnanosecond Response.
IEEE Journal of Solid-State Circuits 4, Vol. 3, December 1968, pp. 365–373.

[15] Glasser L. A:
A UV Write-Enabled PROM.
Chapel Hill Conf. on VLSI
Rockville, MD: Computer Science Press 1985, pp. 61–65.

[16] Goodman A. M:
Electron Hall Effect in Silicon Dioxide.
Physical Review A 164, 1967, pp. A1145–A1150.

[17] Hebb D. O:
The Organization of Behavior.
New York: Wiley 1949. Exerpt printed in [4].

[18] Hellstrom B. J., Kanal L. N:
Knapsack Packing Networks.
IEEE Trans. Neural Networks 2, March 1992, pp. 302–307.

[19] Holler M., Tam S., Castro H., Benson R:
An Electrically Trainable Artifical Neural Network (ETANN) with 10240 "Floating Gate" Synapses.
Proc. Int. Joint Conf. Neural Networks 1989, Vol. II, pp. 191–196.

[20] Hong Z., Melchior H:
Four-Quadrant CMOS Analogue Multiplier.
Electronics Letters 24, Vol. 20, 22nd November 1984, pp. 1015–1016.

[21] Hong Z., Melchior H:
Analogue Four-Quadrant CMOS Multiplier with Resistors.
Electronics Letters 12, Vol. 21, 6th June 1985, pp. 531–532.

[22] Hopfield J. J:
Neural Networks and Physical Systems with Emergent Collective Computational Abilities.
Proc. National Academy of Sciences 79, 1982, pp. 2554–2558. Reprinted in [4].

[23] Hopfield J. J:
Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons.
Proc. National Academy of Sciences 81, 1984, pp. 3088–3092. Reprinted in [4].

[24] Hopfield J. J., Tank D. W:
"Neural" Computation of Decisions in Optimization Problems.
Biological Cybernetics 52, 1985, pp. 141–152.

[25] Ingman D., Merlis Y:
Local Minimum Escape Using Thermodynamic Properties of Neural Networks.
Neural Networks, 1991, pp. 395–404.

[26] James W:
Psychology (Briefer Course).
New York: Holt 1890. Exerpt printed in [4].

[27] Kerns D. A., Tanner J. E., Sivilotti M. A., Luo J:
CMOS UV-Writable Non-Volatile Analog Storage.
Proc. Advanced Research in VLSI Int. Conf., Santa Cruz, CA 1991, pp. 245–261.

[28] Kerns D. A:
Experiments in Very Large-Scale Analog Computation
Ph.D. thesis, California Institute of Technology, 1993.

[29] Khachab N. I., Ismail M:
MOS Multiplier/Divider Cell for Analogue VLSI.
Electronics Letters 23, Vol. 25, 9th November 1989, pp. 1550–1552.

[30] Khachab N. I., Ismail M:
Nonlinear CMOS Analog Cell for VLSI Signal and Information Processing.
IEEE Journal of Solid-State Circuits 11, Vol. 26, November 1991, pp. 1689–1699.

[31] Kim C. W., Park S. B:
New Four-Quadrant CMOS Analogue Multiplier.
Electronics Letters 24, Vol. 23, 19th November 1987, pp. 1268–1270.

[32] Kim Y. H., Park S. B:
Four-Quadrant CMOS Analogue Multiplier.
Electronics Letters 7, Vol. 28, 26th March 1992, pp. 649–650.

[33] Kuh A., Dickinson B. W:
Information Capacity of Associative Memories.
IEEE Trans. Information Theory 1, January 1989, pp. 59–68.

[34] Larsen M:
Integrert Optoelektronisk Mottagersystem.
Cand. Scient. thesis, Institute of Informatics, University of Oslo, December 1992.

[35] Lee B. W., Sheu B. J:
Modified Hopfield Neural Networks for Retrieving the Optimal Solution.
IEEE Trans. Neural Networks 1, January 1991, pp. 136–142.

[36] Little W. A:
The Existence of Persistent States in the Brain.
Mathematical Biosciences 19, 1974, pp. 101–120.

[37] Maher M.-A:
New UV-Memory Writing Scheme.
Unpublished

[38] Masa P., Hoen K., Wallinga H:
20 Million Patterns per Second Analog CMOS Neural Network Pattern Classifier.
Proc. 11th European Conf. Circuit Theory and Design 1993, Vol. I, pp. 497–502.

[39] McClelland J. L., Rumelhart D. E:
Parallell Distributed Processing – Explorations in the Microstructure of Cognition. Vol. II: Psychological and Biological Models.
MIT Press 1986.
See also [47]

[40] McEliece R. J., Posner E. C., Rodemich E. R., Venkatesh S. S:
The Capacity of the Hopfield Associative Memory.
IEEE Trans. Information Theory 4, July 1987, pp. 461–482.

[41] Mead C. A:
Analog VLSI and Neural Systems.
Addison Wesley 1989.

[42] Mekkaoui A., Jespers P:
Four Quadrant Multiplier for Neural Networks.
Electronics Letters 4, Vol. 27, 14th February 1991, pp. 320–322.

[43] Meta-Software:
HSPICE User's Manual.
Meta-Software, Inc., 1991.

[44] Ngolediage J. E., Dlay S. S., Gorgui-Naguib R. N:
CMOS Phase Detector and Four Quadrant Multiplier for Implementation in Analogue Neural Networks.
Electronics Letters 12, Vol. 28, 4th June 1992, pp. 1142–1143.

[45] Personnaz L., Guyon I., Dreyfus G:
Information Storage and Retrieval in Spin-Glass Like Neural Networks.
Journal de Physique Lettres 46, 15 Avril 1985, pp. 359–365.

[46] Prados D. L., Kak S. C:
Neural Network Capacity using Delta Rule.
Electronics Letters 3, Vol. 25, 2nd February 1989, pp. 197–199.

[47] Rumelhart D. E., McClelland J. L:
Parallell Distributed Processing – Explorations in the Microstructure of Cognition. Vol. I: Foundations.
MIT Press 1986.
See also [39]

[48] Sakurai S., Ismail M:
High Frequency Wide Range CMOS Analogue Multiplier.
Electronics Letters 19, Vol. 28, 19th November 1992, pp. 2228–2229.

[49] Schneider C., Card C:
Analogue CMOS Hebbian Synapses.
Electronics Letters 9, Vol. 27, 25th April 1991, pp. 785–786.

[50] Silva-Martinez J., Sanchez-Sinencio E:
Analogue OTA Multiplier without Input Voltage Swing Restrictions, and Temperature Compensated.
Electronics Letters 11, Vol. 22, 22nd May 1986, pp. 599–600.

[51] Soelberg K:
Kontinuerlige "backpropagation" nett i analog VLSI.
Cand. Scient. thesis, Institute of Informatics, University of Oslo, September 1992.

[52] Soo D. C., Meyer R. G:
A Four-Quadrant NMOS Analog Multiplier.
IEEE Journal of Solid-State Circuits 6, Vol. 17, December 1982, pp. 1174–1178.

[53] Streetman B. G:
Solid State Electronic Devices.
Prentice Hall International 1990.

[54] Sussmann H. J:
On the Number of Memories that can be Perfectly Stored in a Neural Net with Hebbian Weights.
IEEE Trans. Information Theory 1, January 1989, pp. 174–179.

[55] Takefuji Y., Lee K. C:
Artificial Neural Networks for Four-Coloring Map Problems and K*-Colorability Problems.*
IEEE Trans. Circuits and Systems 3, March 1991, pp. 326–333.

[56] Tank D. W., Hopfield J. J:
Concentrating Information in Time: Analog Neural Networks with Applications to Speech Recognition Problems.

[57] Tank D. W., Hopfield J. J:
Simple "Neural" Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit.
IEEE Trans. Circuits and Systems 5, May 1986, pp. 533–541.

[58] Van den Bout D. E., Miller T. K:
A Travelling Salesman Objective Function That Works.
IEEE Proceedings ICNN'88, Vol. II 1988, pp. 299–303.

[59] Van den Bout D. E., Miller T. K:
Graph Partitioning Using Annealed Neural Networks.
IEEE Trans. Neural Networks 2, June 1990, pp. 192–203.

[60] Venkatesh S. V., Psaltis D:
Linear and Logarithmic Capacities in Associative Neural Networks.
IEEE Trans. Information Theory 3, May 1989, pp. 558–568.

[61] Wang Z:
A CMOS Four-Quadrant Analog Miltiplier with Single-Ended Voltage Output and Improved Temperature Performance.
IEEE Journal of Solid-State Circuits 9, Vol. 26, September 1991, pp. 1293–1301.

[62] Watts L., Kerns D. A., Lyon R. F., Mead C. A:
Improved Implementation of the Silicon Cochlea.
IEEE Journal of Solid-State Circuits 5, Vol. 27, May 1992, pp. 692–700.

[63] Weste N. H. E., Eshraghian K:
Principles of CMOS VLSI Design.
Addison Wesley 1988.

[64] Williams R:
Photo Emission of Electrons from Silicon into Silicon Dioxide.
Physical Review A 140, 1965, pp. A569–A575.

[65] Wilson G. V., Pawley G. S:
On the Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank.
Biological Cybernetics 58, 1988, pp. 63–70.

[66] Xu X., Tsai W. T:
Effective Neural Algorithms for the Travelling Salesman Problem.
Neural Networks, 1991, pp. 193–205.

Appendix A

Results from the net

A.1 Retrieval of one vector

The following pages contain the measurements for all 64 test vectors during retrieval of the stored vector $[1, 1, -1, -1, 1, 1]$.

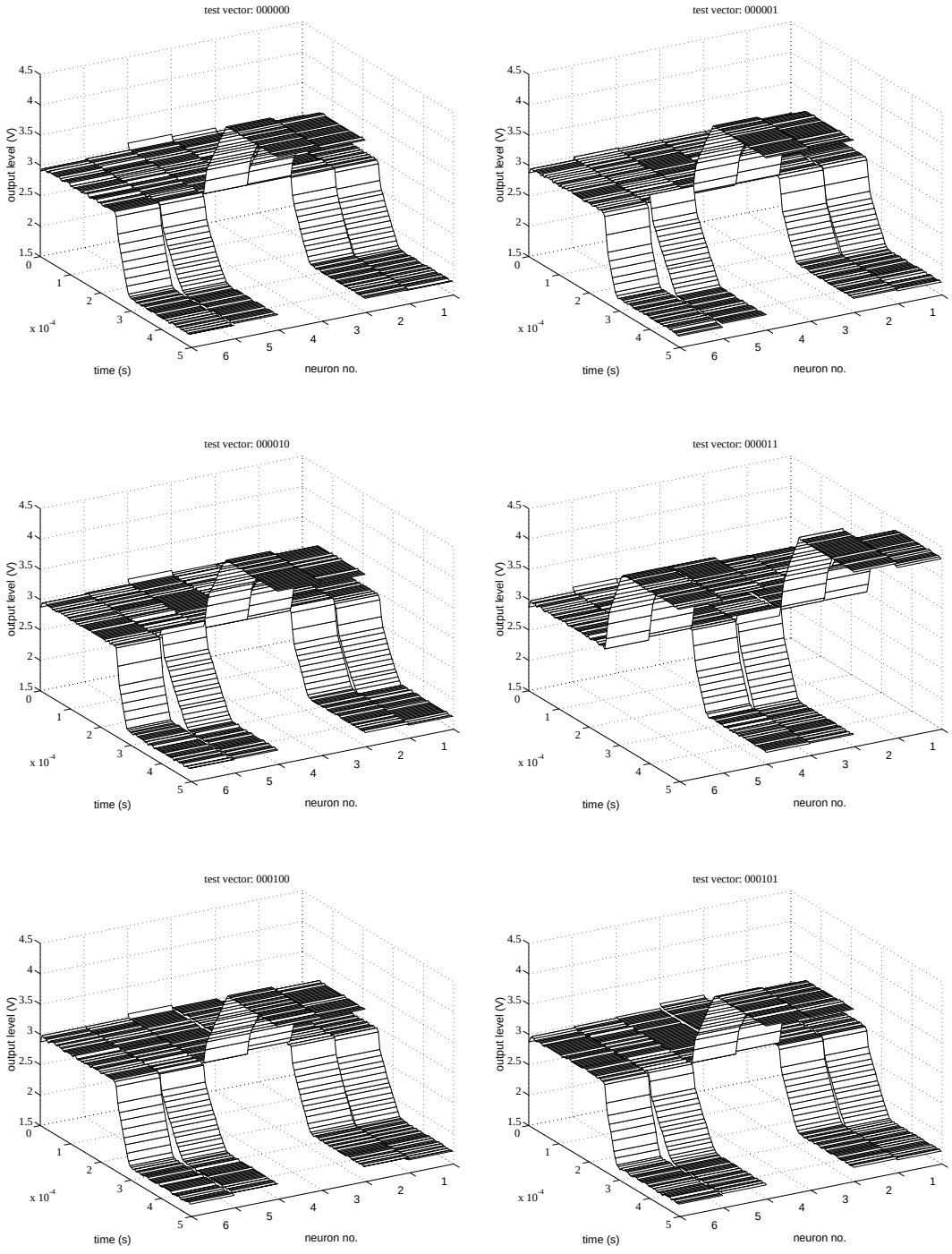


Figure A.1: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

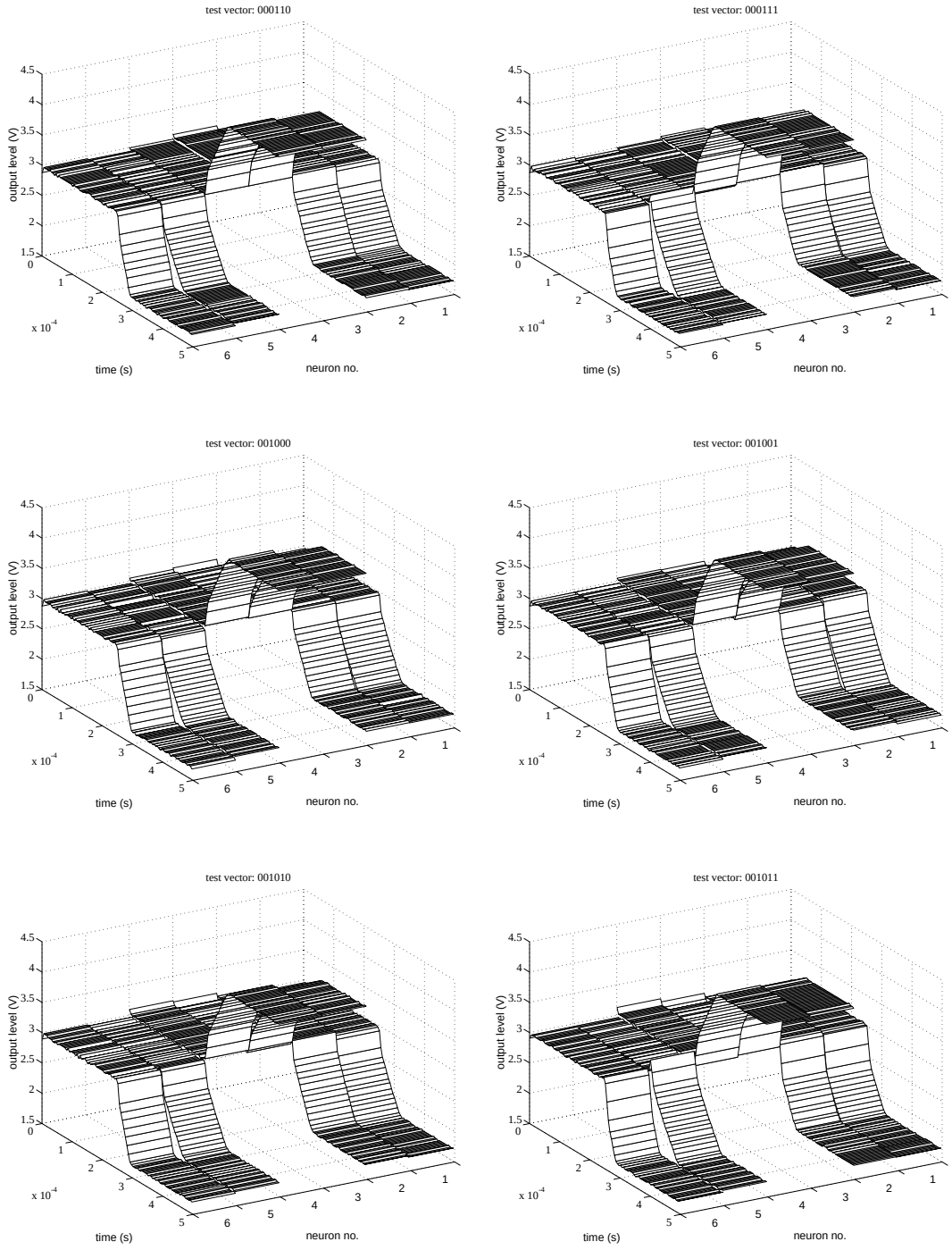


Figure A.2: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

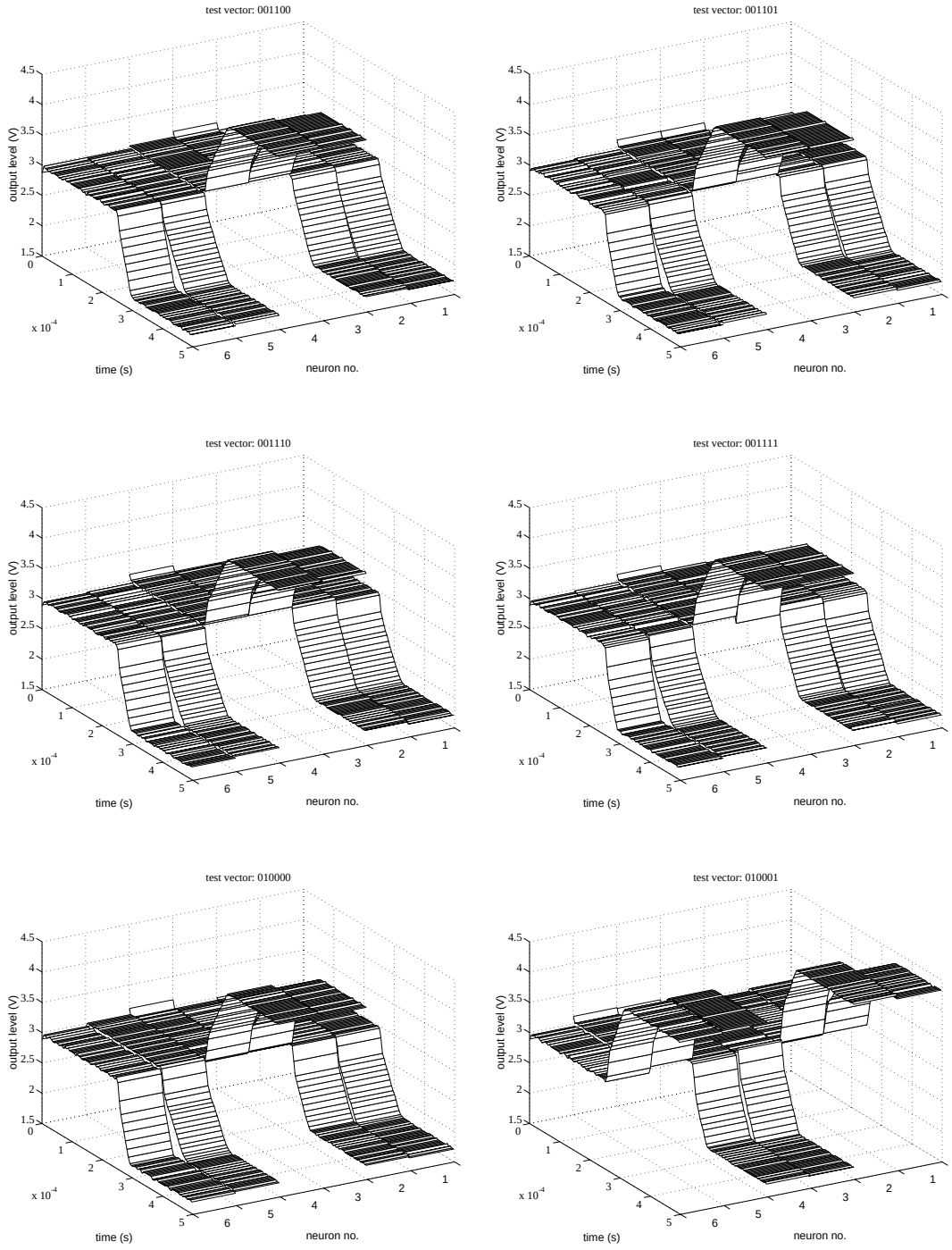


Figure A.3: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

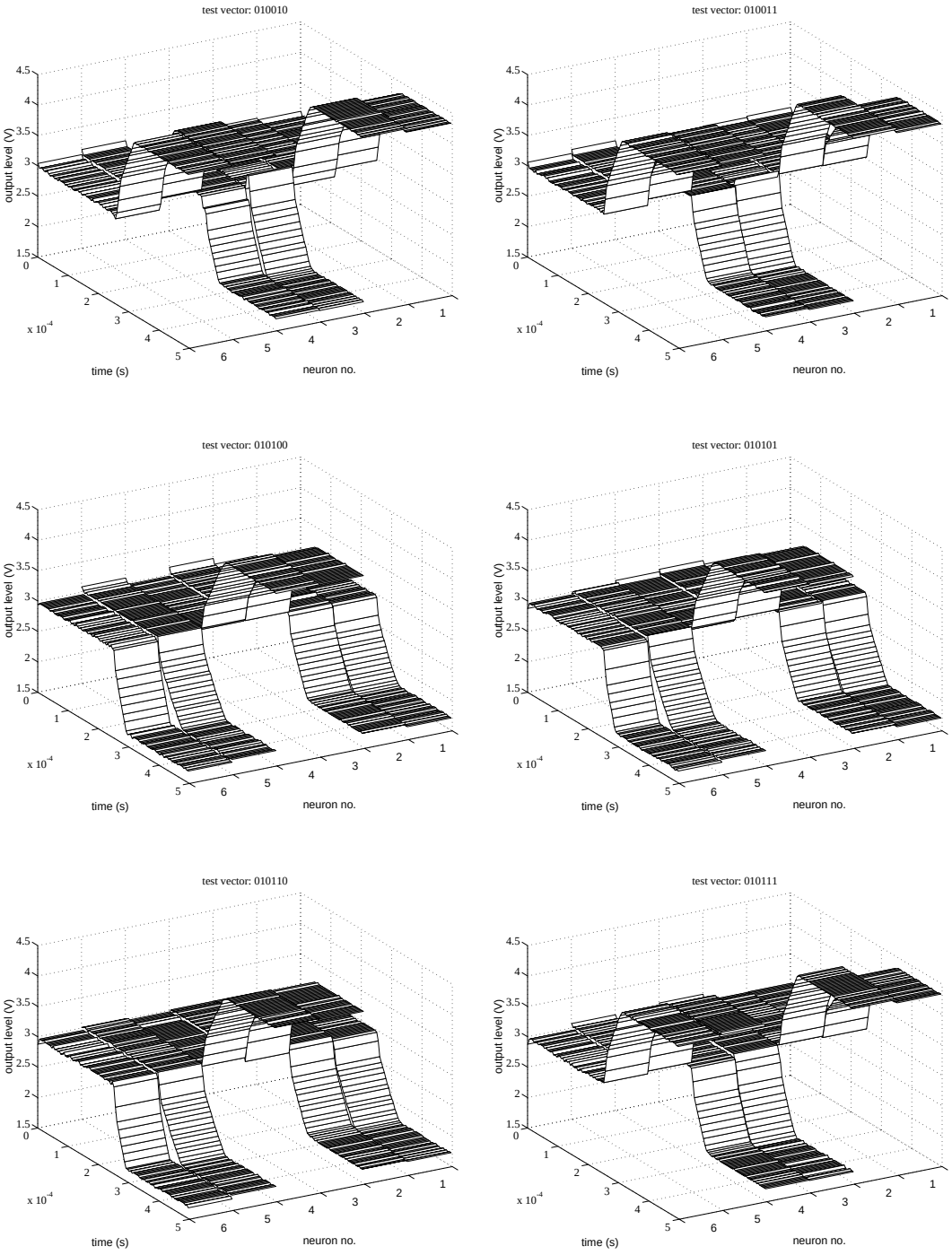


Figure A.4: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

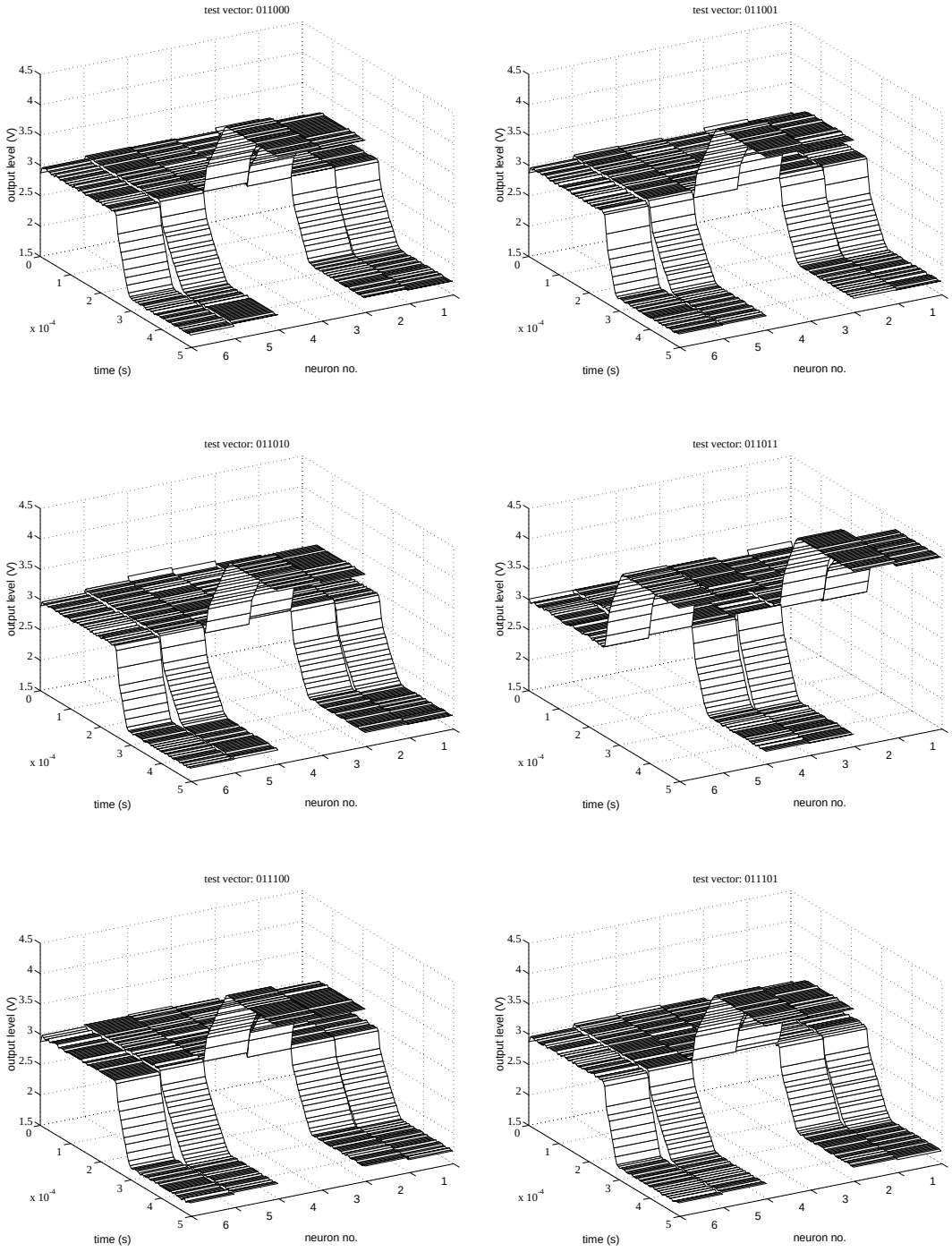


Figure A.5: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

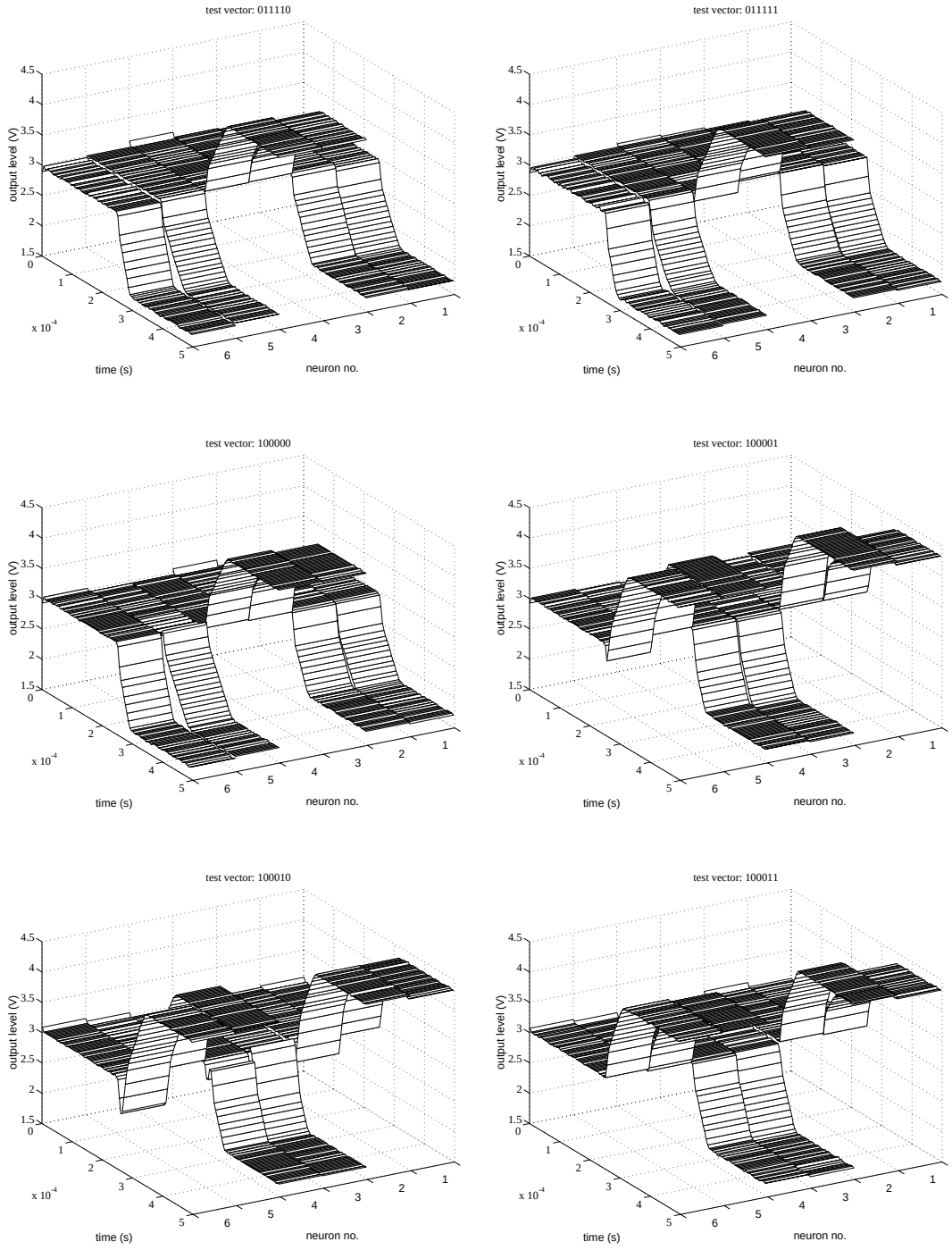


Figure A.6: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

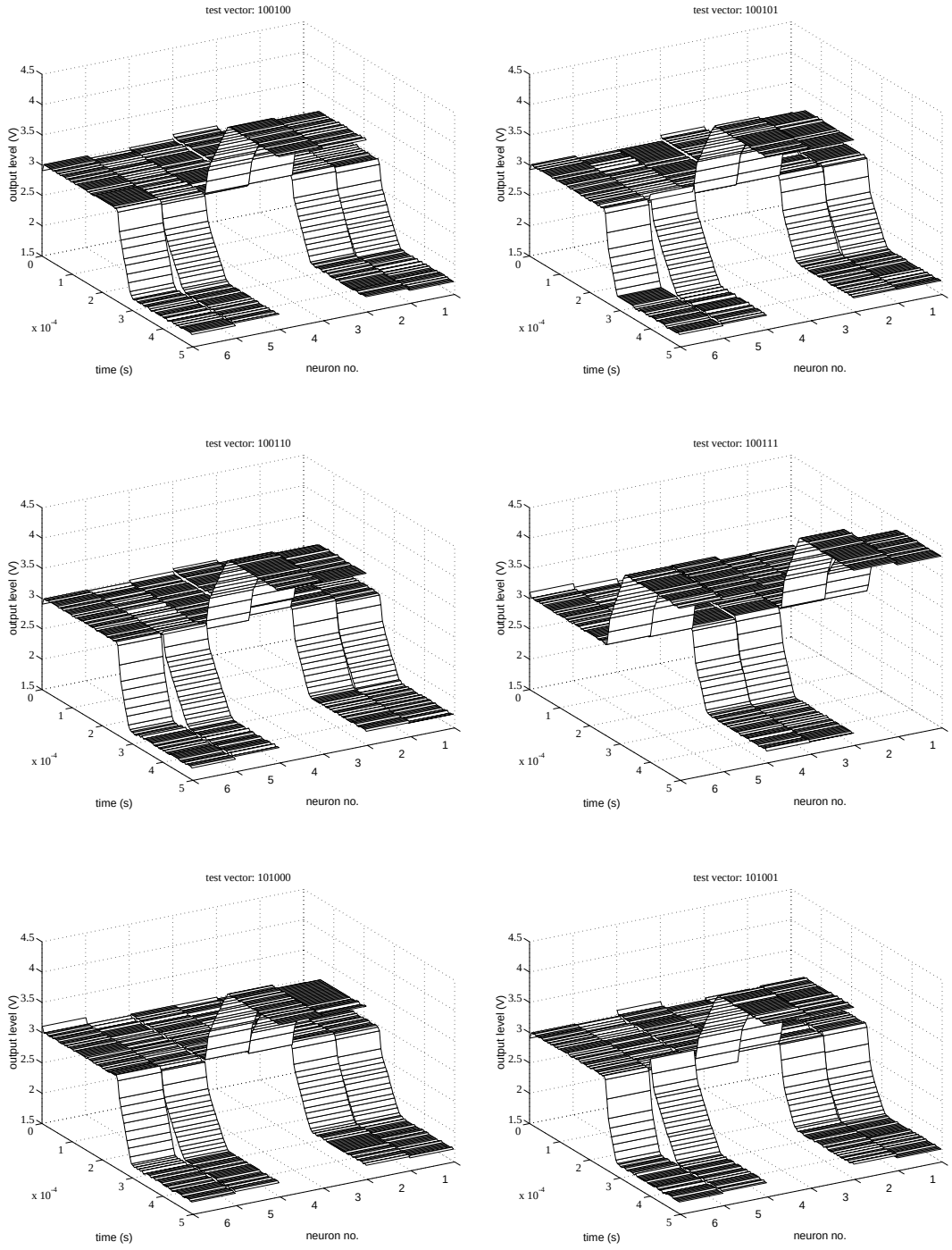


Figure A.7: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

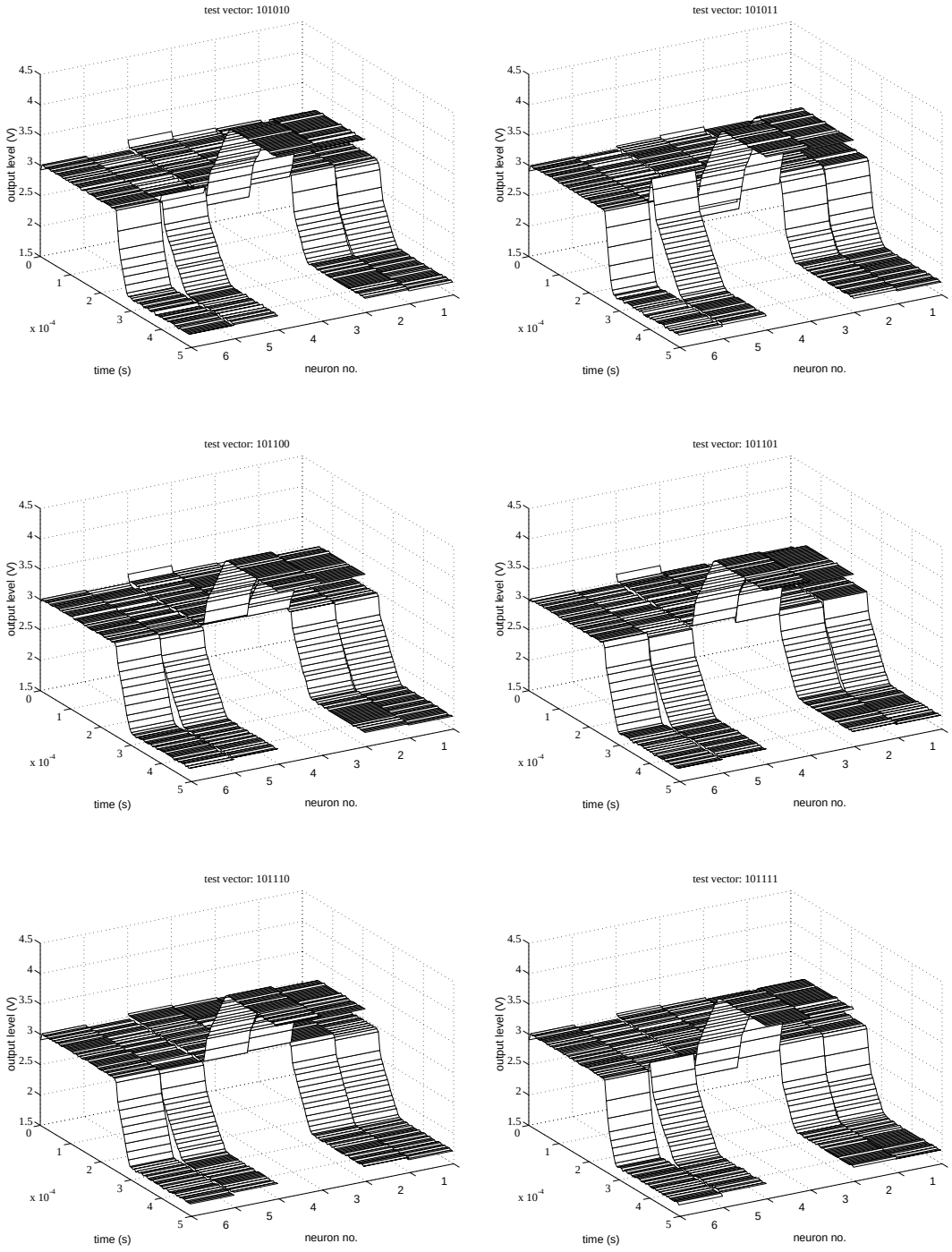


Figure A.8: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

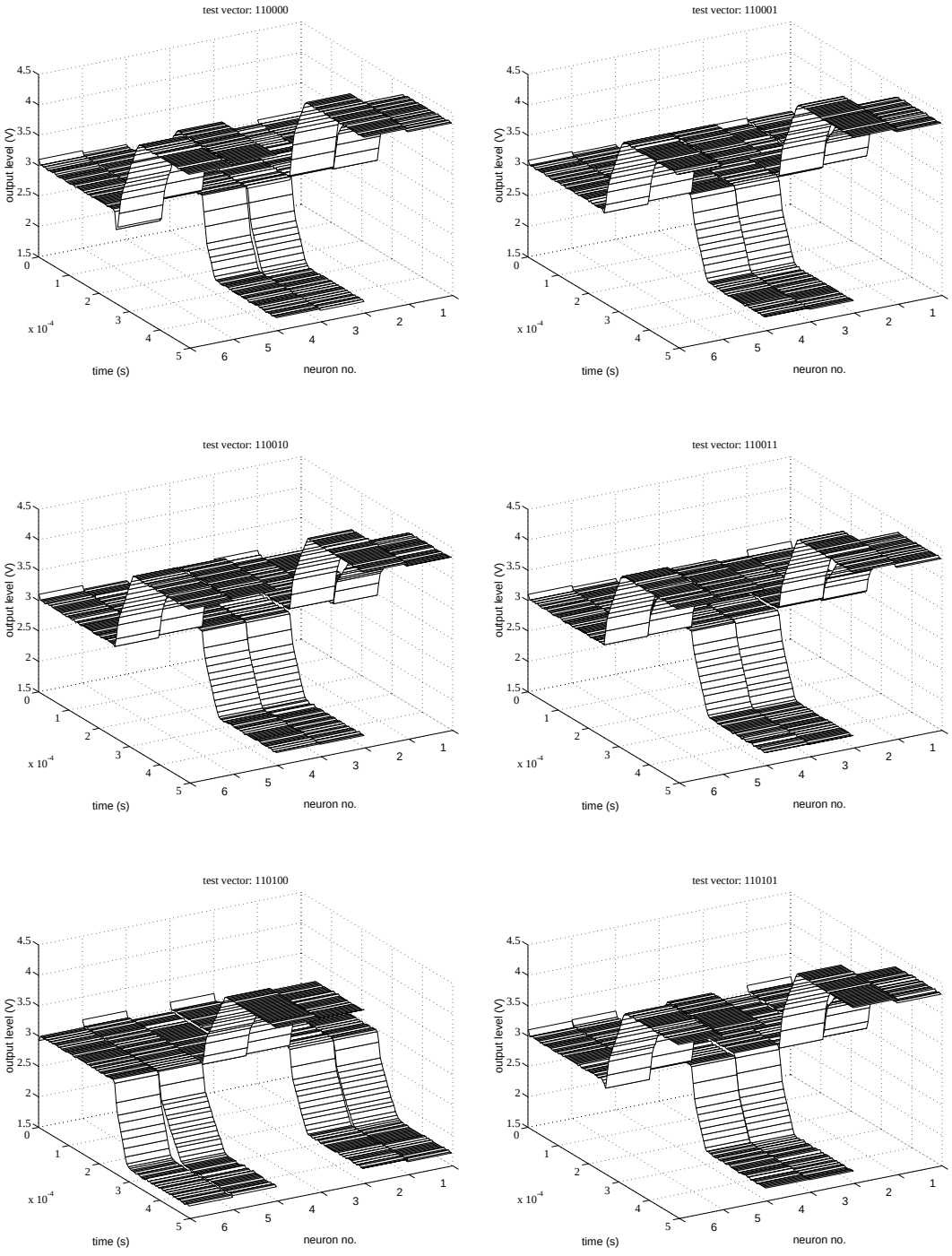


Figure A.9: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

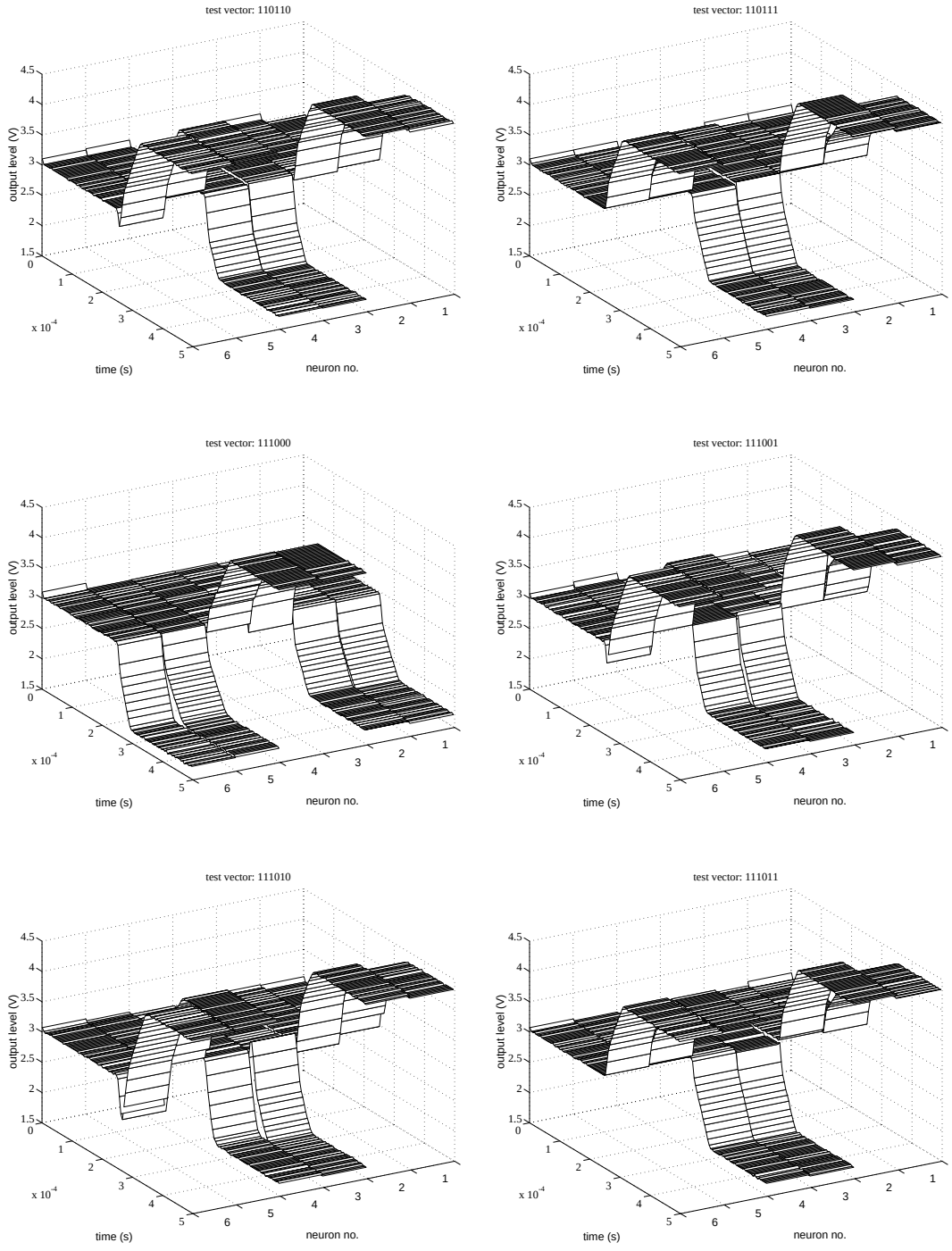


Figure A.10: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

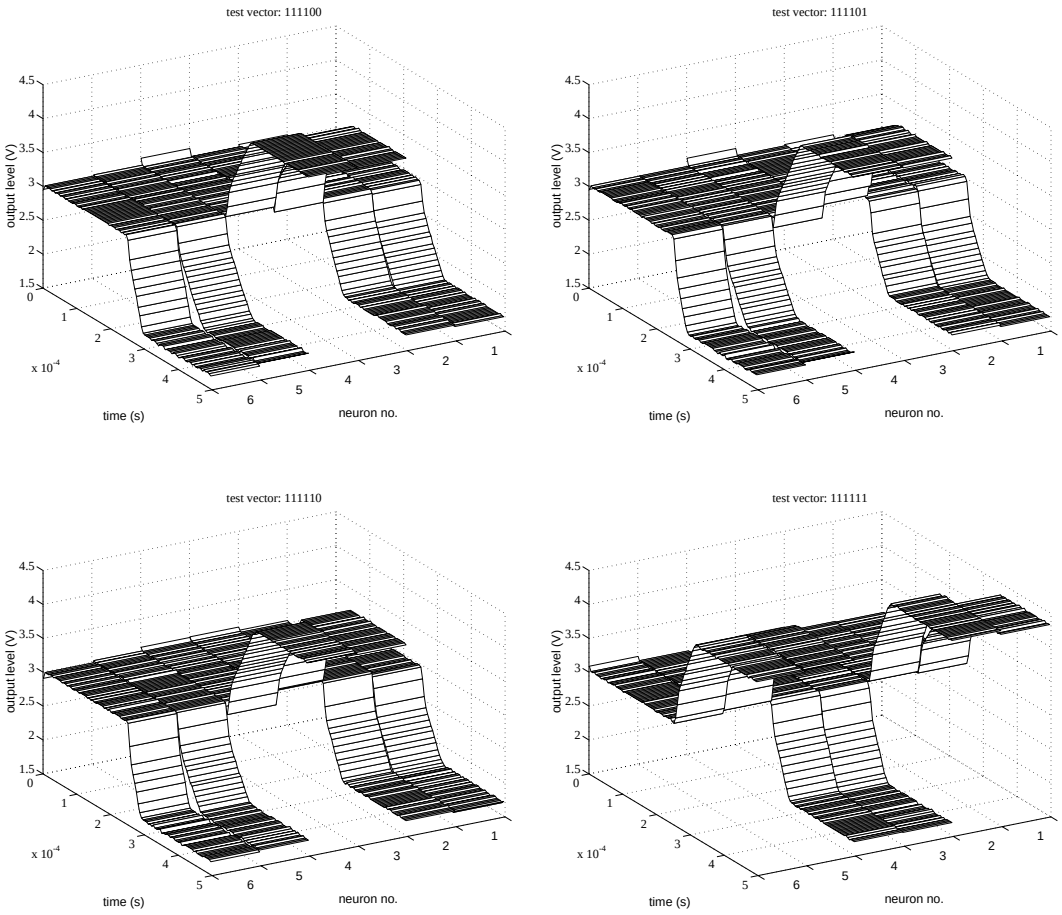


Figure A.11: *Measured results of retrieval of the stored vector [-1,-1,1,1,-1,-1].*

A.2 Retrieval of two vectors

The following pages contain the measurements for all 64 test vectors during retrieval of the stored vectors $u = [1, 1, -1, -1, 1, 1]$ and $v = [1, 1, 1, -1, -1, -1]$. This series is special in that it contains one example of the net not converging to a stored vector (test vector #26 $[-1, 1, 1, -1, -1, 1]$). Repeated measurements have not reproduced this type of error. In this particular series the net showed a tendency to pick v above u. The error rate is approximately 29%.

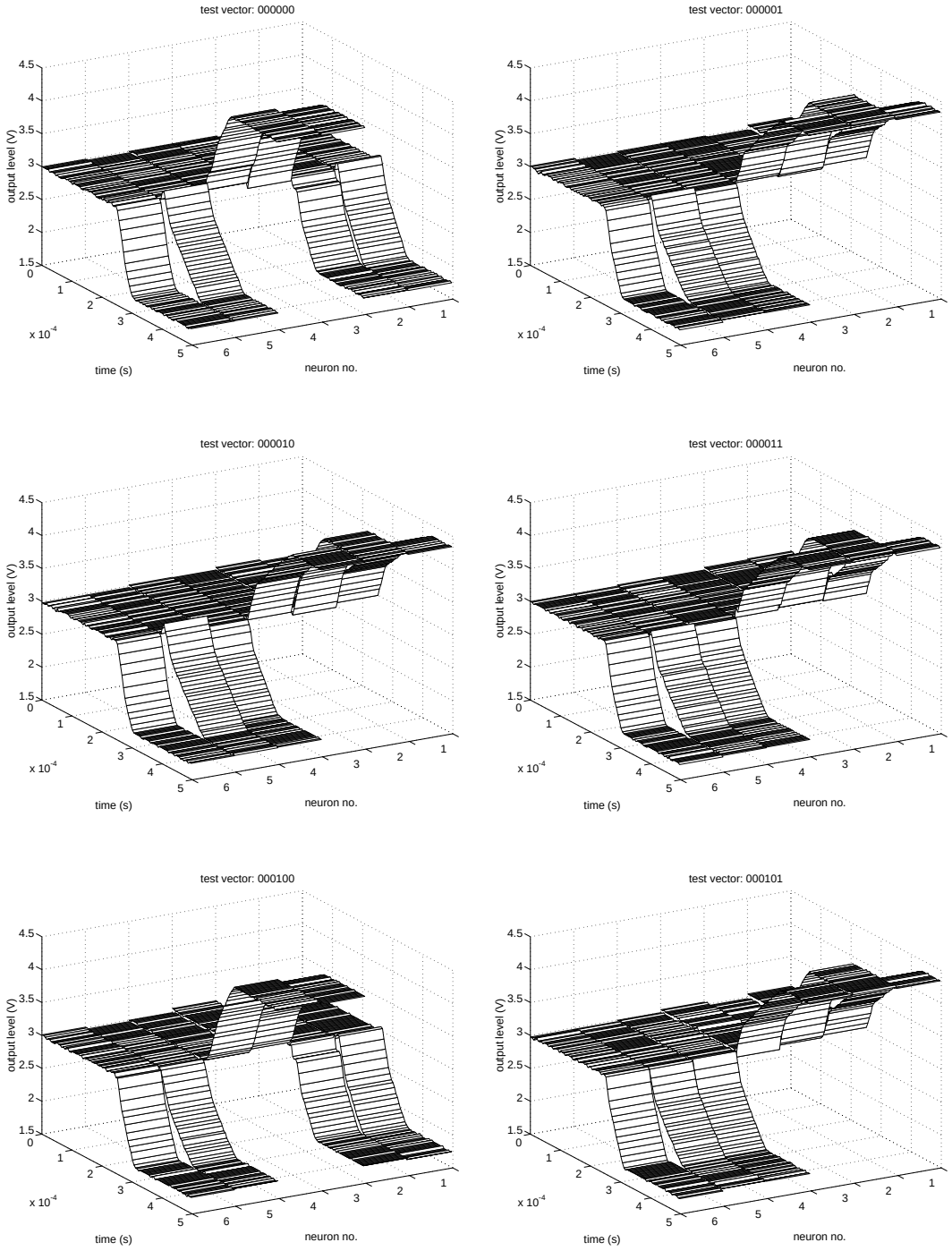


Figure A.12: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 1–6.*

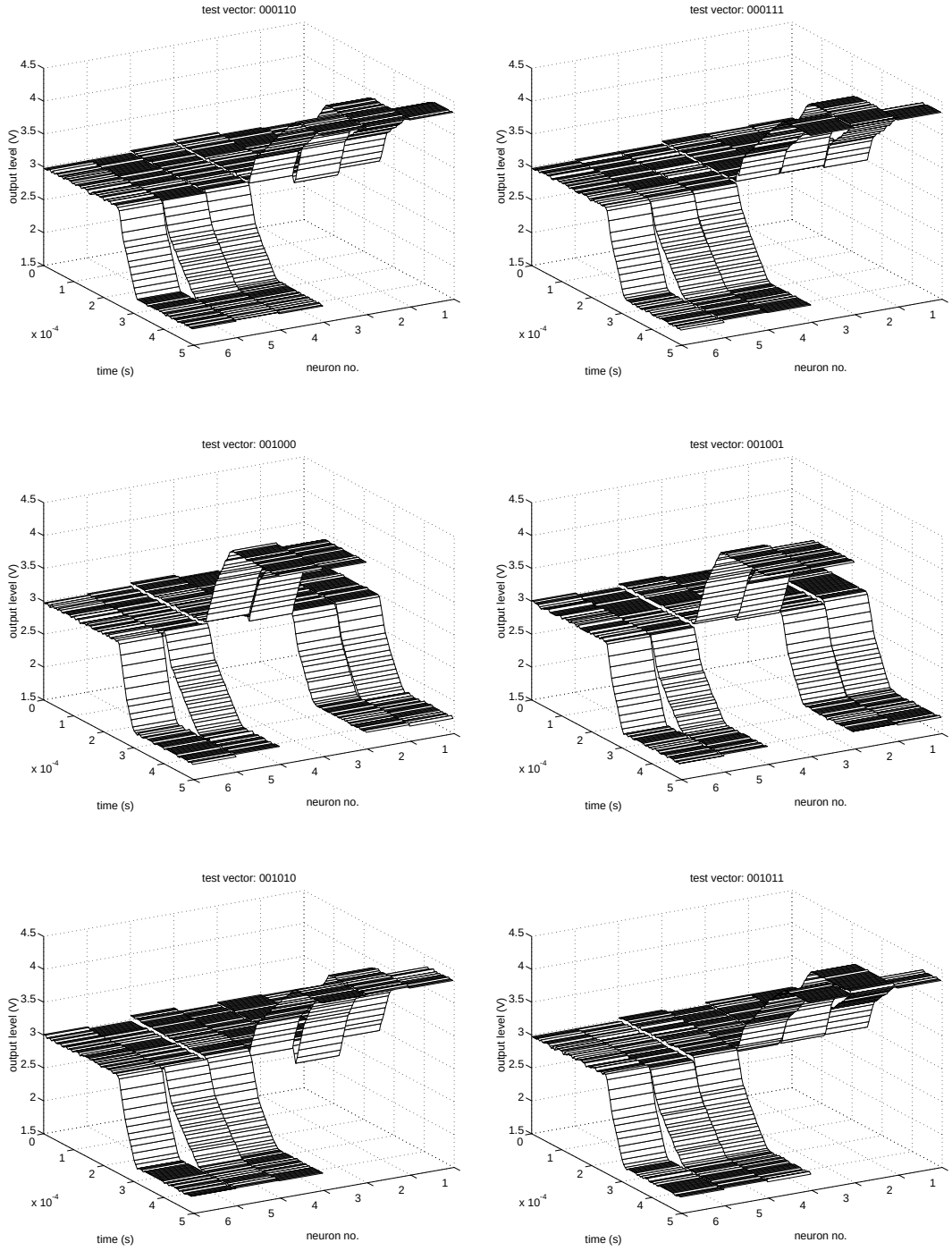


Figure A.13: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 7–12.*

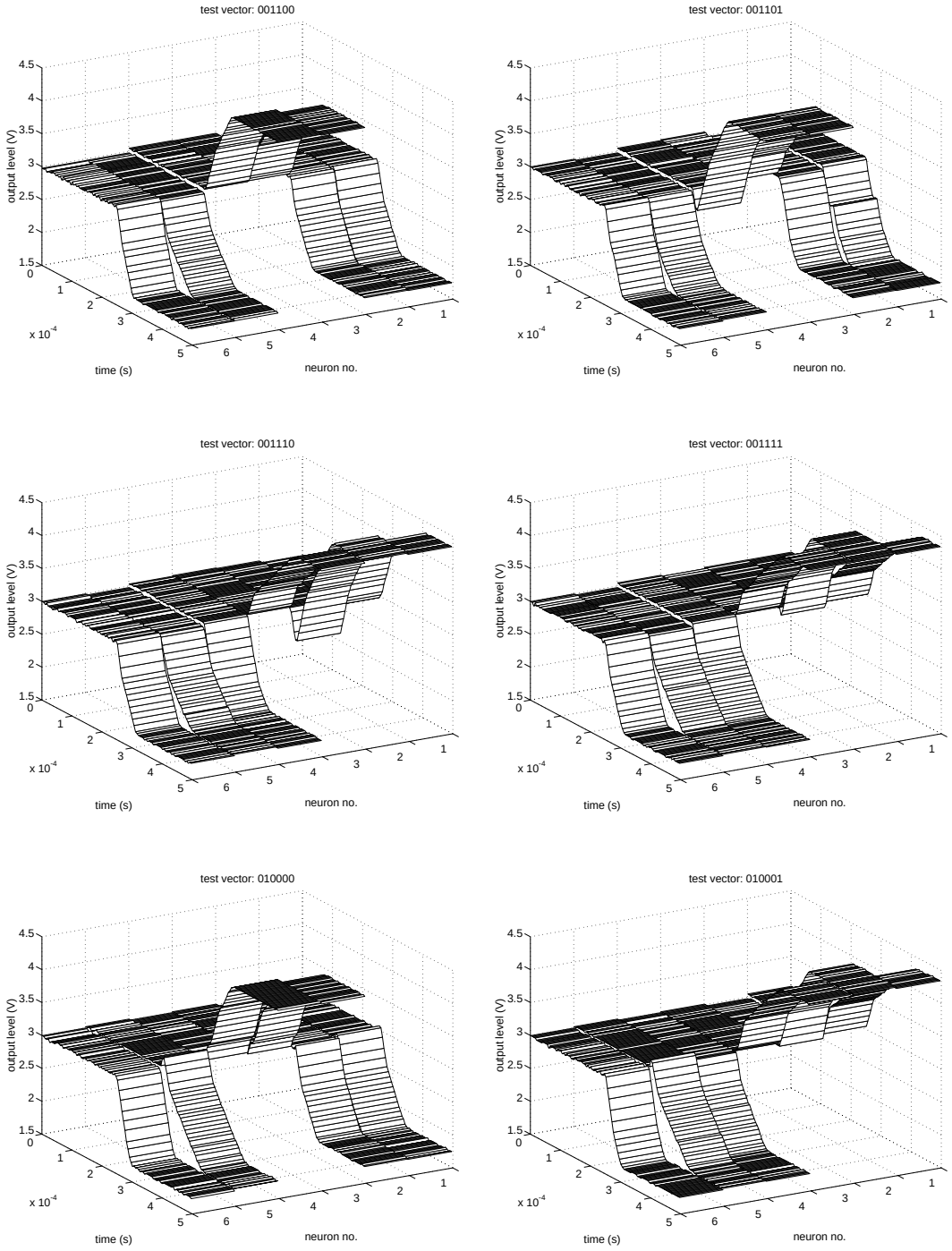


Figure A.14: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 13–18.*

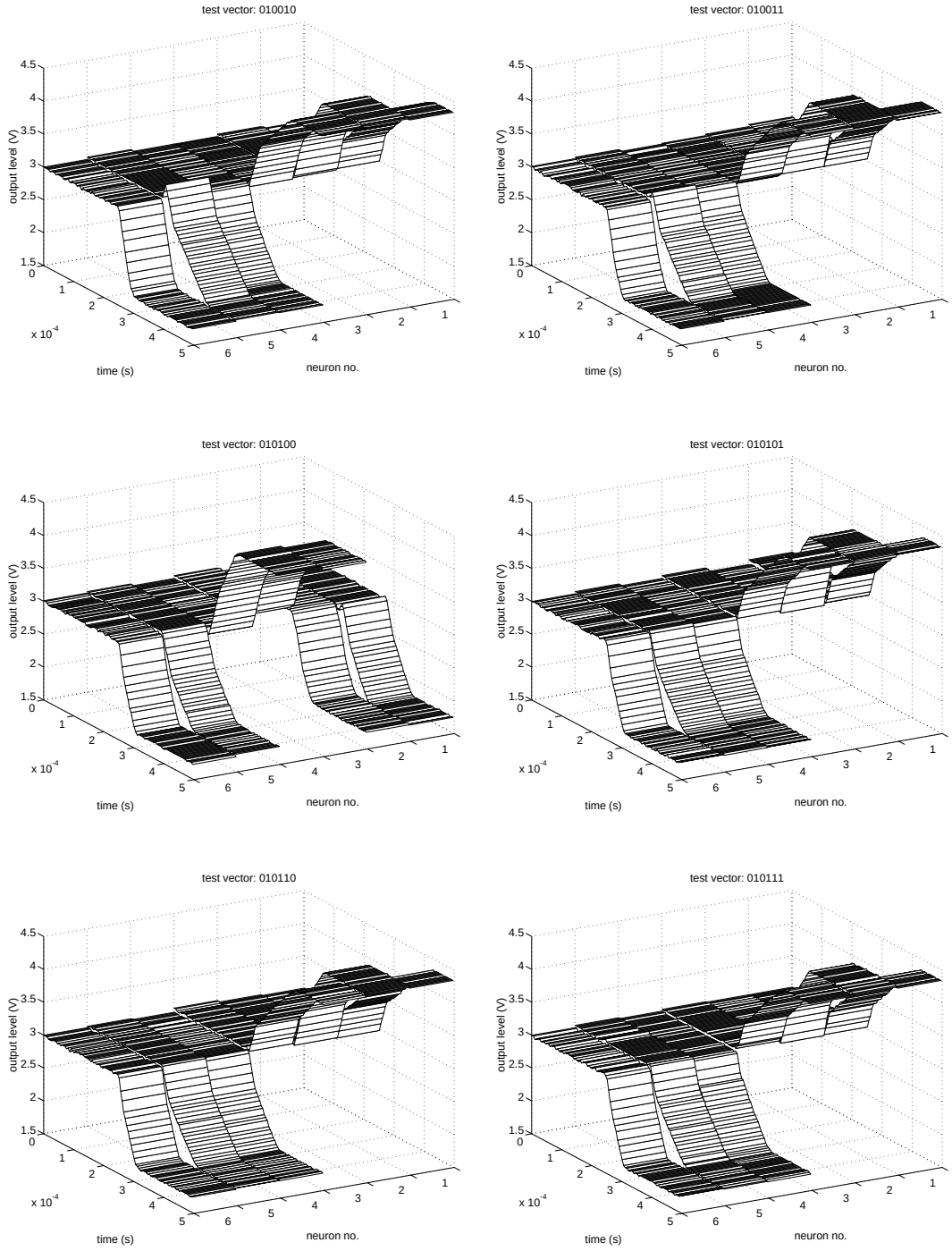


Figure A.15: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 19–24.*

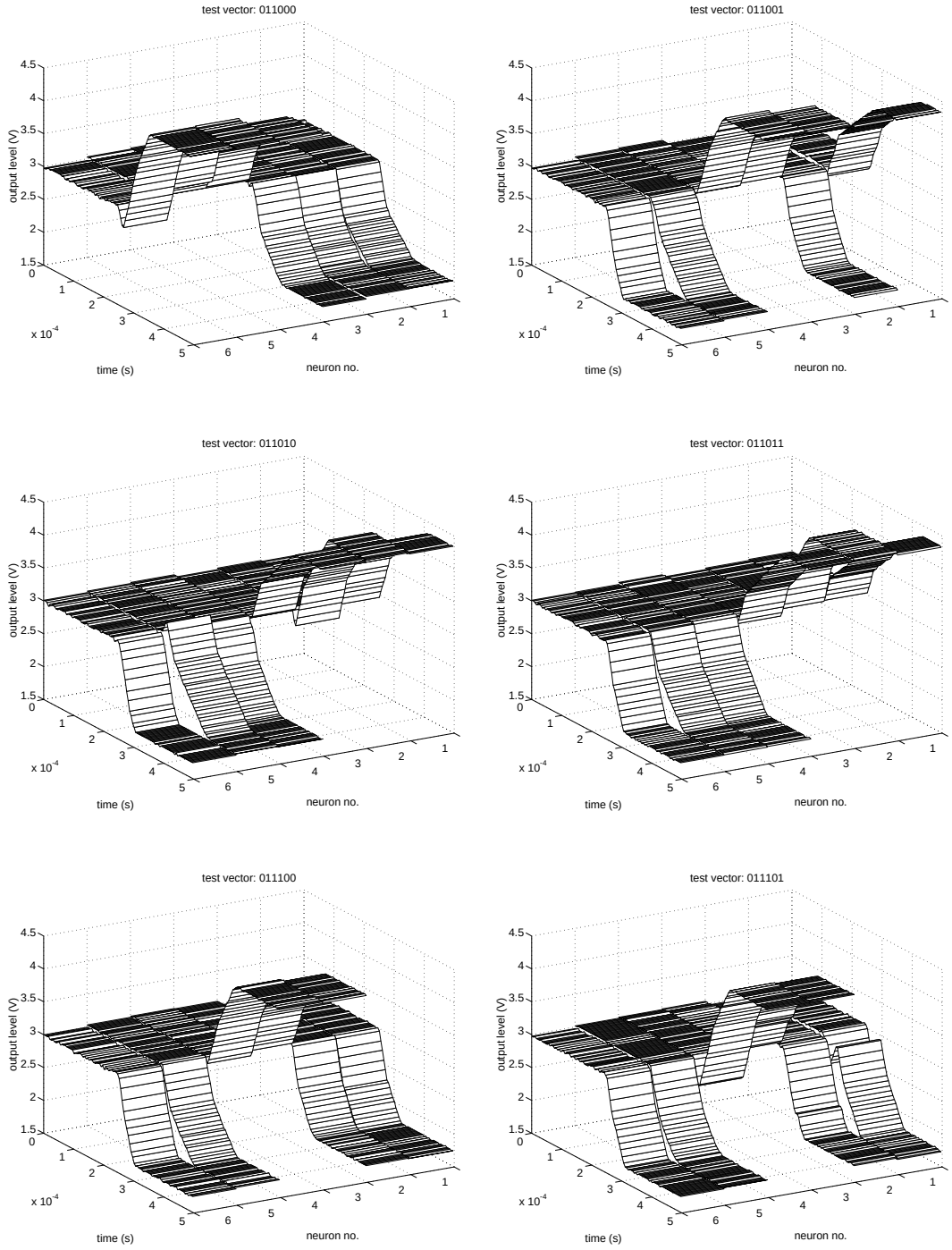


Figure A.16: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 25–30.*

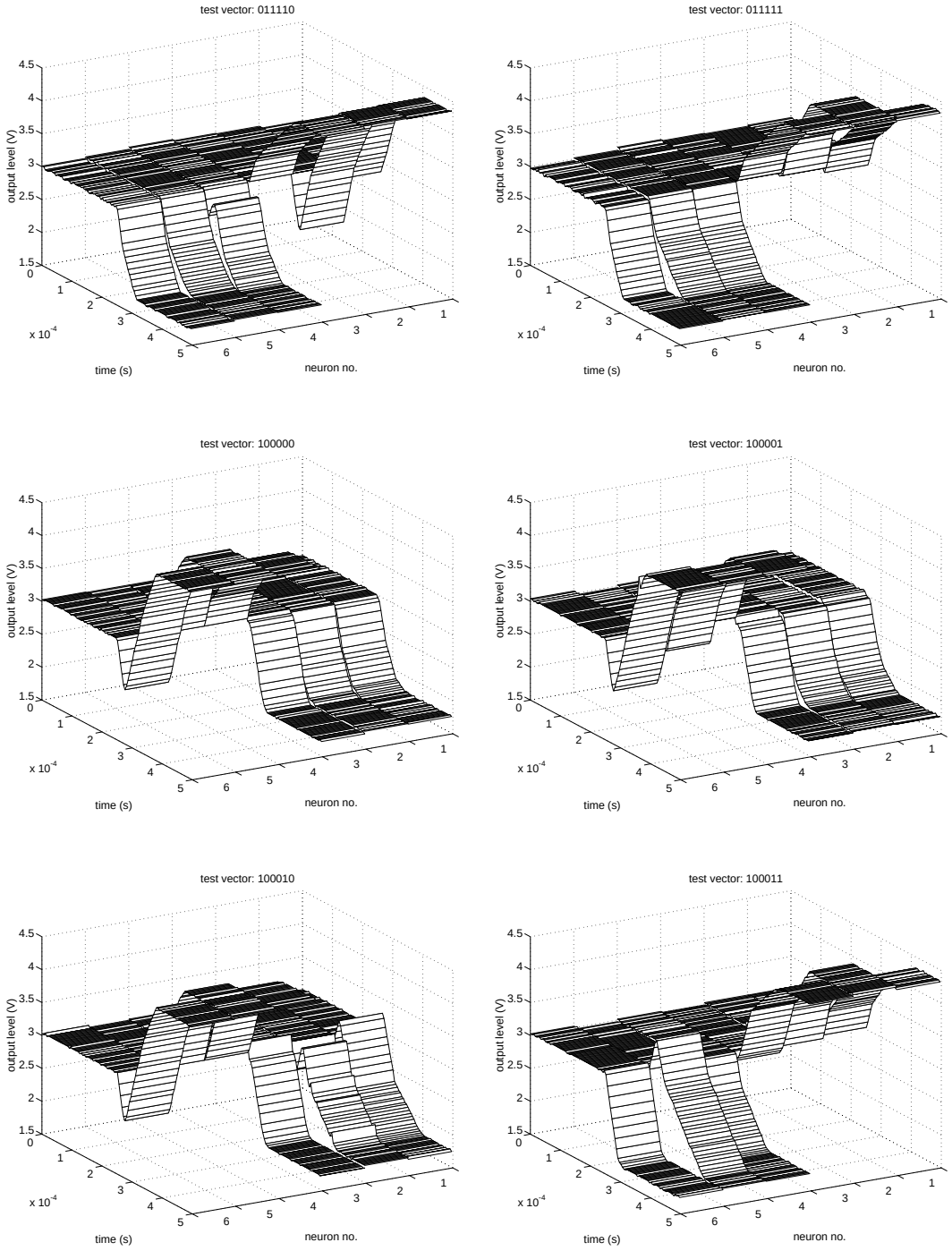


Figure A.17: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 31–36.*

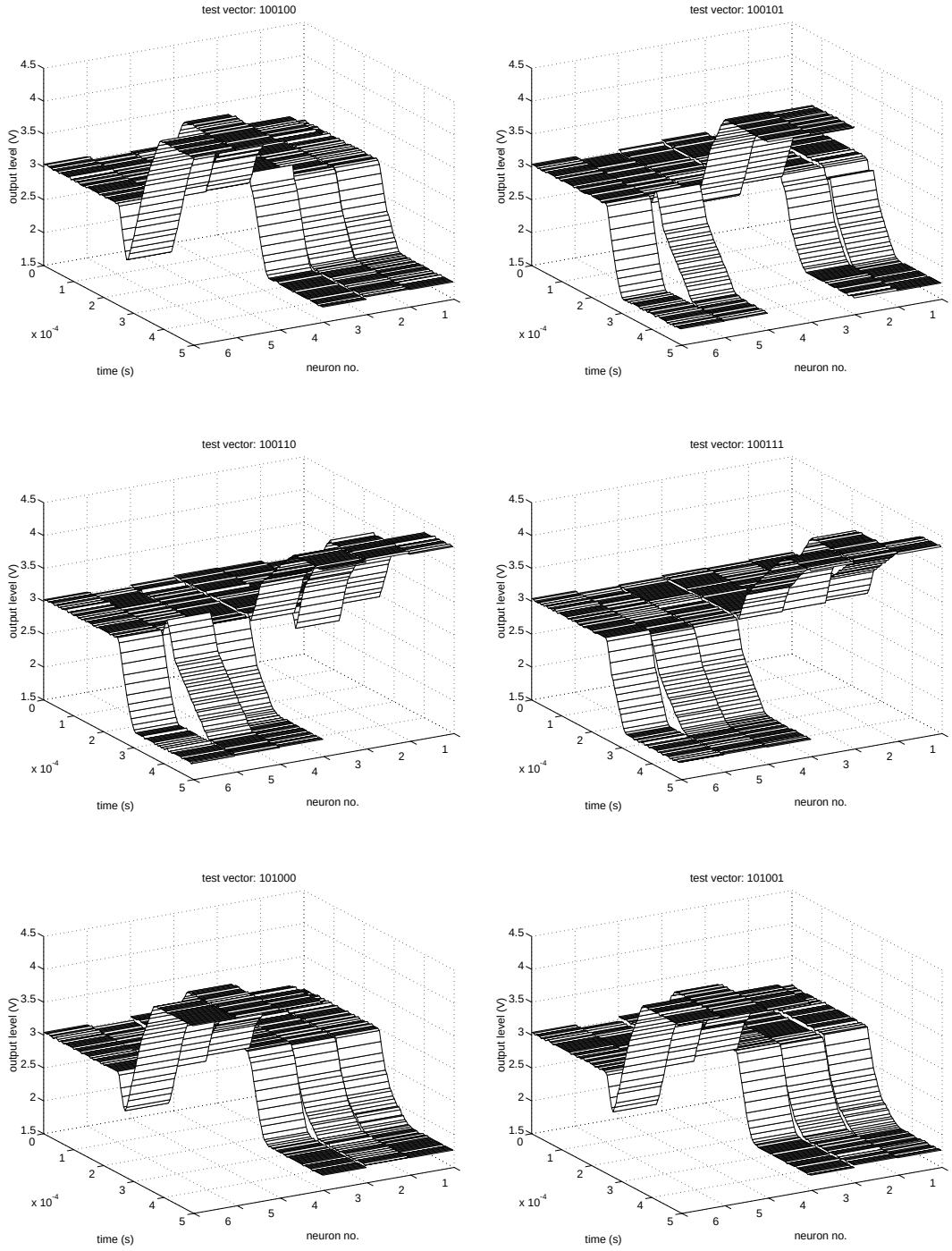


Figure A.18: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 37–42.*

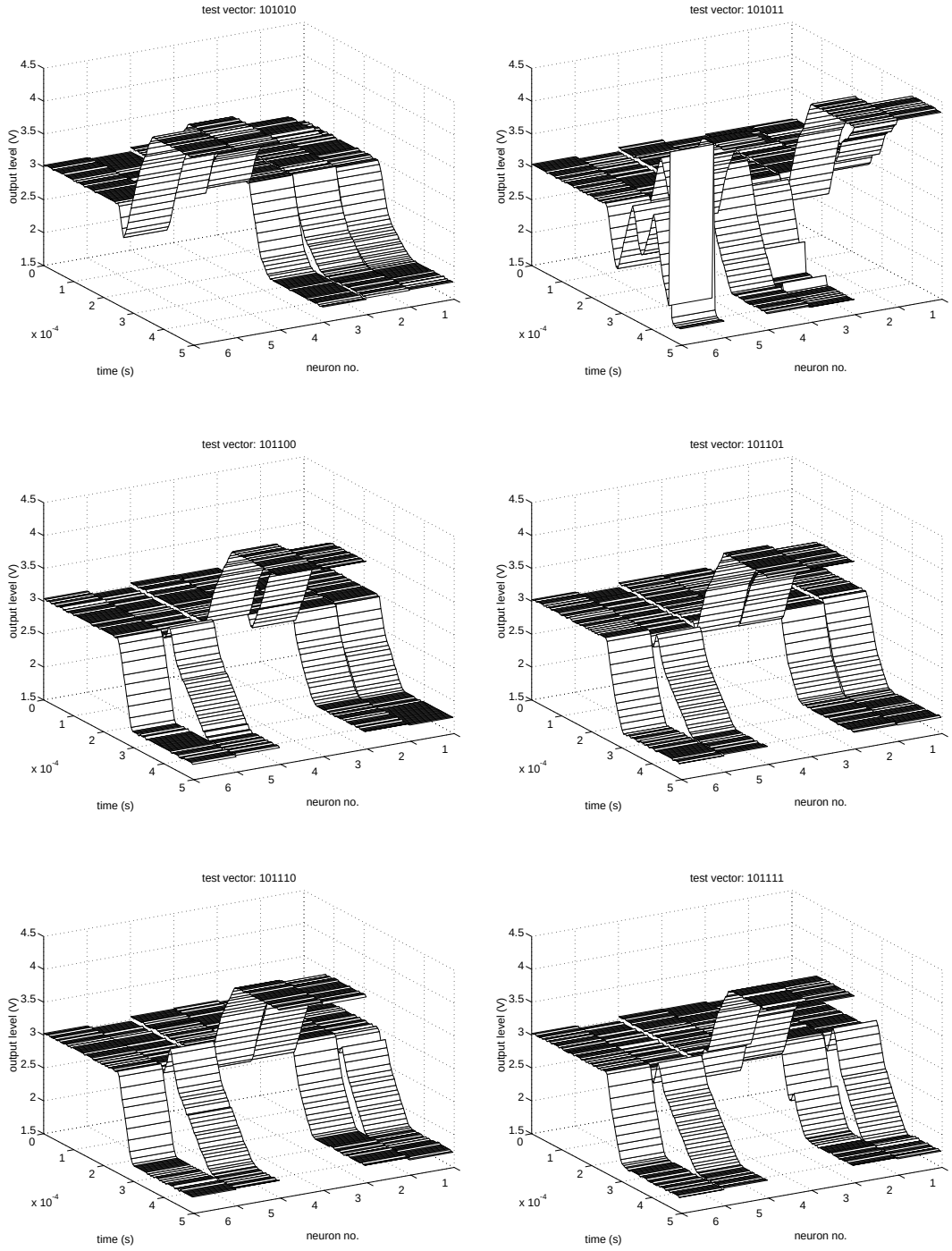


Figure A.19: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 43–48.*

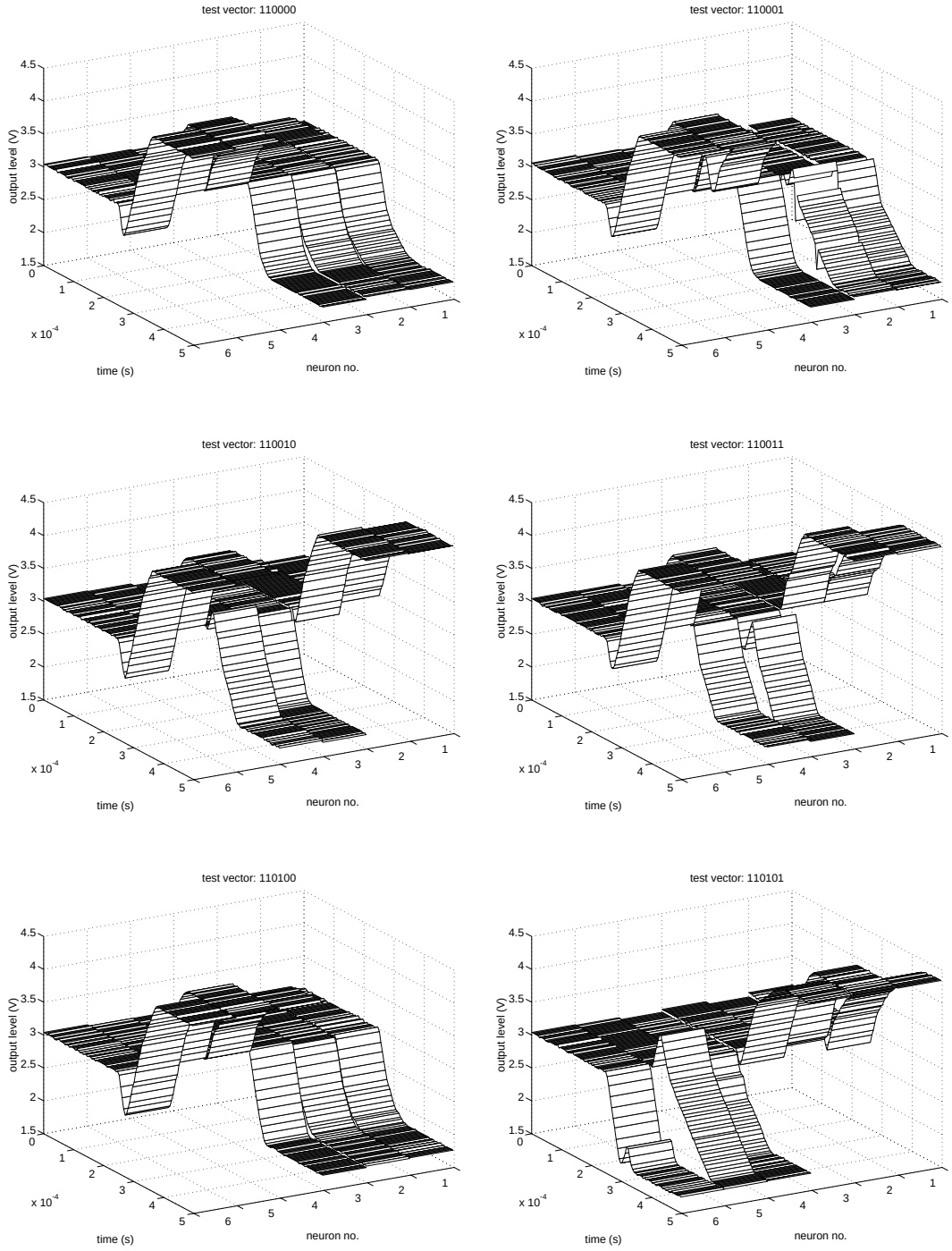


Figure A.20: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 49–54.*

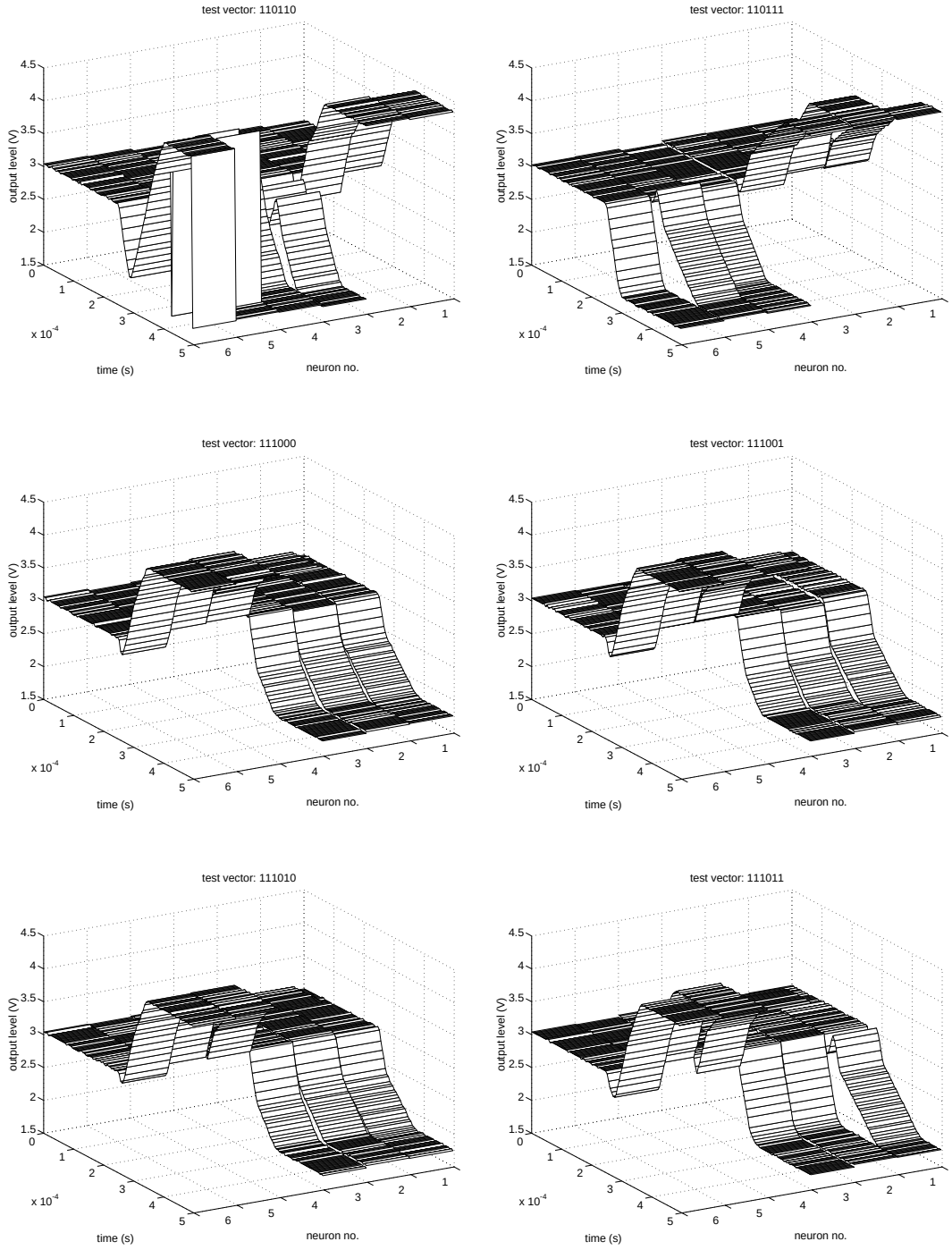


Figure A.21: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 55–60.*

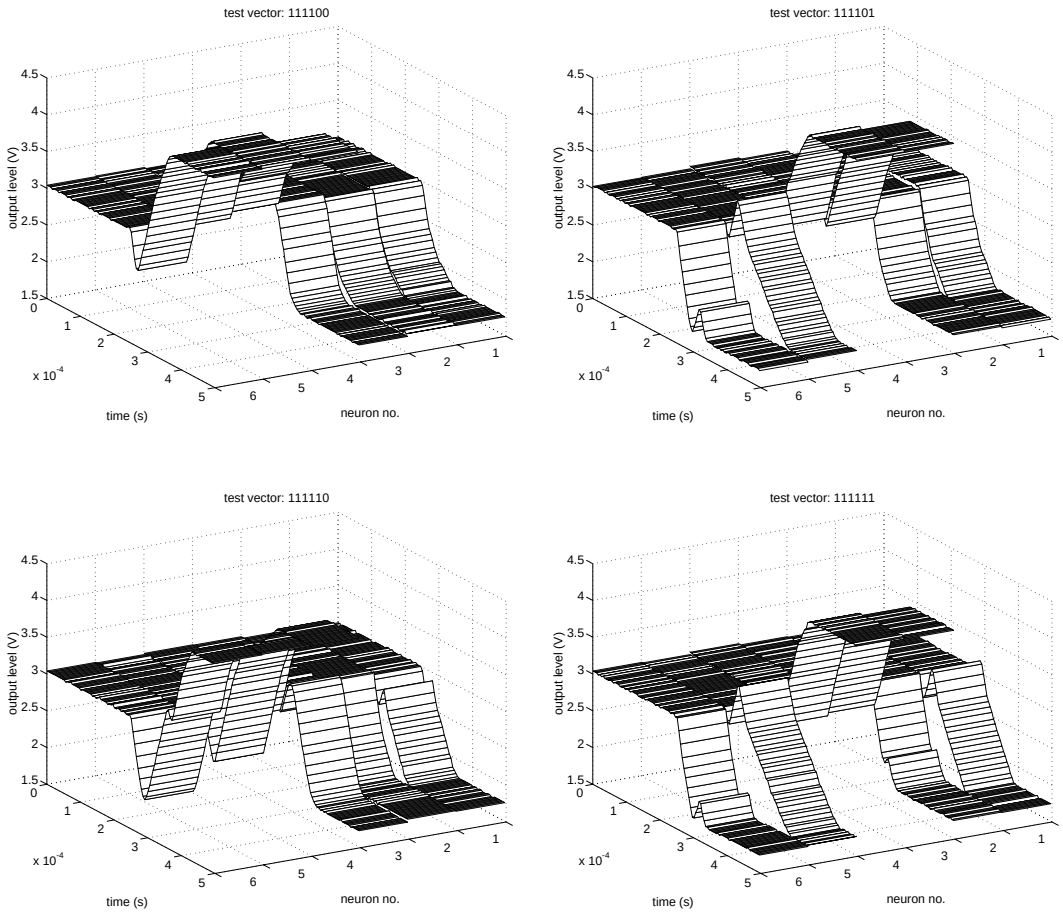


Figure A.22: *Measured results of retrieval of the stored vectors [1,1,1,-1,-1,-1] and [1,1,-1,-1,1,1]. Test vectors 61–64.*

Appendix B

Simulation

During the work for this thesis I have had access to two simulators, the anaLOG simulator written by Massimo Sivilotti at Caltech, and HSPICE h9007 from Meta-Software. anaLOG has a beautiful interactive user interface, and a circuit idea can usually be simulated within minutes. Also anaLOG was developed especially for subthreshold simulation. All building blocks for this thesis were developed using anaLOG. The drawback is that there is a fairly low limit on the number of transistors that may be simulated at one time, and it is difficult to include user defined functions. Since the point about neural nets is that the whole is greater than the sum of the parts, I needed a simulator able to handle larger circuits. The only option was SPICE.

At that time I did not have access to transistor parameters for HSPICE from Orbit, the chip vendor used for my chips. I found some parameters for Berkely SPICE, level 3 (the BSIM model) and adapted them to HSPICE's BSIM model (level 13). I have later learned that the BSIM model is not considered very accurate.

The parameters came with no values expressing body-effect. These were set by me at what I felt were reasonable values. Also, when simulating the drain characteristsics, I noticed that Early effects where a lot less prominent than in real life. But at least I got an exponential gate characteristic, which was what I wanted.

SPICE was never intended for subthreshold simulation, which is very noticeable. One of the main problems is that HSPICE places a conductance in parallel with a MOS transistor to enhance convergence properties [43, pp. 2-33]. To get reasonable subthreshold currents, this conductance must be reduced (the GMINDC and GMIN parameters), which leads to convergence problems. A lot of time was spent just convincing HSPICE to run a simulation without increasing this conductance, and thereby invalidating the results. Being unsure of whether a bad simulation result is due to the circuit or the simulator is not conductive to good work progression. But, from comparing the simulations to measurements on real circuits, I feel that the end product was not too bad.

Compared to anaLOG, the user interface to HSPICE (batch processed netlists written in emacs) is really bad. The one property of HSPICE's batch processing that I appreciated, was the ability to do parametric simulation, which I took to the limit.

B.1 Transistor parameters

```
* BSIM1 PARAMETERS FROM RUN N11B AT ORBIT *
.model orbit_pm1_du1 pmos level=13
+ vfb0 = -0.186475 lvfb = 0.0557384 wvfb = 0.163435
+ phi0 = 0.672582 lphi = 0 wphi = -7.55995e-24
+ k1 = 0.515094 lk1 = 0.00168647 wk1 = -0.0187785
+ k2 = -0.0376313 lk2 = 0.058798 wk2 = -0.0324532
```

```
+ eta0 = -0.00523608 leta = 0.0347844 weta = 0.0104252
+ muz = 210.626 dl0 = 0.786519 dw0 = 0.303918
+ u00 = 0.109169 lu0 = 0.0514795 wu0 = -0.0674901
+ u1 = 0.0013656 lu1 = 0.1894 wu1 = -0.052859
+ x2m = 9.58349 lx2m = -5.6521 wx2m = 7.17721
+ x2e = 0.000451863 lx2e = -0.00522038 wx2e = 0.00160425
+ x3e = 0.000142046 lx3e = -0.00222081 wx3e = -0.00123393
+ x2u0 = 0.0056698 lx2u0 = -0.00411785 wx2u0 = 0.00513007
+ x2u1 = -0.00122417 lx2u1 = 0.00729297 wx2u1 = 0.00169588
+ mus = 203.544 lms = 93.1863 wms = 22.3543
+ x2ms = 6.84986 lx2ms = 1.97217 wx2ms = 8.59866
+ x3ms = -1.01036 lx3ms = 9.33157 wx3ms = -2.80944
+ x3u1 = -0.0172891 lx3u1 = 0.00428714 wx3u1 = 0.00703118
+ toxm = 0.0391 tempm = 27 vddm = 5
+ cgdom = 5.20965e-10 cgsom = 5.20965e-10 cgbom = 7.39323e-10
+ xpart = 1
+ n0 = 1.5 ln0 = 0 wn0 = 0
+ nb0 = 0 lnb = 0 wnb = 0
+ nd0 = 0 lnd = 0 wnd = 0
+ rshm = 29.6 cjm = 9.84e-05 cjsw = 4.455e-10
+ ijs = 1e-08 pj = 0.8 pjw = 0.8
+ mj0 = 0.7693 mjw = 0.2906 wdf = 0
+ ds = 0
*
.model orbit_nm1_du1 nmos level=13
+ vfb0 = -0.916252 lvfb = 0.287307 wvfb = -0.356457
+ phi0 = 0.767306 lphi = 1.03147e-24 wphi = -4.04719e-24
+ k1 = 1.34407 lk1 = -0.412991 wk1 = 0.706422
+ k2 = 0.28553 lk2 = -0.0630758 wk2 = 0.0193497
+ eta0 = -0.0111122 leta = 0.0351177 weta = 0.0040575
+ muz = 564.027 dl0 = 0.546736 dw0 = 0.116027
+ u00 = 0.0543847 lu0 = 0.0352634 wu0 = -0.0446096
+ u1 = 0.0802473 lu1 = 0.844767 wu1 = -0.575386
+ x2m = 6.53591 lx2m = -4.53033 wx2m = 59.5132
+ x2e = -0.0023869 lx2e = -0.00672737 wx2e = -0.00314927
+ x3e = 0.000674031 lx3e = -0.00218632 wx3e = -0.00487129
+ x2u0 = 0.000959215 lx2u0 = 0.000154626 wx2u0 = 0.0215285
+ x2u1 = -0.0173526 lx2u1 = 0.0231123 wx2u1 = 0.0473201
+ mus = 756.807 lms = 354.525 wms = -197.496
+ x2ms = -2.29945 lx2ms = 24.2432 wx2ms = 111.247
+ x3ms = 9.32926 lx3ms = 88.5792 wx3ms = -64.0163
+ x3u1 = 0.0173669 lx3u1 = 0.0878708 wx3u1 = -0.0773039
+ toxm = 0.0391 tempm = 27 vddm = 5
+ cgdom = 3.6214e-10 cgsom = 3.6214e-10 cgbom = 6.9311e-10
+ xpart = 1
+ n0 = 1.5 ln0 = 0 wn0 = 0
+ nb0 = 0 lnb = 0 wnb = 0
+ nd0 = 0 lnd = 0 wnd = 0
+ rshm = 29.6 cjm = 9.84e-05 cjsw = 4.455e-10
+ ijs = 1e-08 pj = 0.8 pjw = 0.8
+ mj0 = 0.7693 mjw = 0.2906 wdf = 0
+ ds = 0
```

B.2 Library file

```
*
.lib amplifiers
*
.subckt inverter    in  out
M1  out  in    Vdd  Vdd  orbit_pm1_du1 W=18u L=6u
M2  out  in    0    0    orbit_nm1_du1 W=6u L=6u
.ends inverter
$
.subckt WR_amp   neg pos bias out
M1  3    bias  0    0    orbit_nm1_du1 W=6u L=18u
M2  6    neg   3    0    orbit_nm1_du1 W=6u L=6u
M3  7    pos   3    0    orbit_nm1_du1 W=6u L=6u
M4  6    6     Vdd  Vdd  orbit_pm1_du1 W=6u L=12u
M5  7    7     Vdd  Vdd  orbit_pm1_du1 W=6u L=12u
M6  8    6     Vdd  Vdd  orbit_pm1_du1 W=6u L=12u
M7  out  7     Vdd  Vdd  orbit_pm1_du1 W=6u L=12u
M8  8    8     0    0    orbit_nm1_du1 W=6u L=12u
M9  out  8     0    0    orbit_nm1_du1 W=6u L=12u
.ends WR_amp
*
.subckt i_amp   in ref bias out
M10 out  out  in    0    orbit_nm1_du1 W=6u L=6u
M11 out  out  in    Vdd  orbit_pm1_du1 W=12u L=6u
X1  in   ref  bias  out  WR_amp
.ends i_amp
*
$
.endl amplifiers
*
*
*
.lib multipliers
*
.subckt gilbert_core   inAp inAm inBp inBm bias outA outB
M1  3     bias  0    0    orbit_nm1_du1 W=12u L=24u
M2  6     inAp  3    0    orbit_nm1_du1 W=6u L=4u
M3  7     inAm  3    0    orbit_nm1_du1 W=6u L=4u
M4  outA  inBp  6    0    orbit_nm1_du1 W=6u L=4u
M5  outB  inBm  6    0    orbit_nm1_du1 W=6u L=4u
M6  outA  inBm  7    0    orbit_nm1_du1 W=6u L=4u
M7  outB  inBp  7    0    orbit_nm1_du1 W=6u L=4u
.ends gilbert_core
*
.subckt gilbert_mul   inAp inAm inBp inBm bias out
M7  op   op    8    Vdd  orbit_pm1_du1 W=6u L=12u  $ Benson diode
M8  8    8     Vdd  Vdd  orbit_pm1_du1 W=6u L=12u
M9  out  8     Vdd  Vdd  orbit_pm1_du1 W=6u L=12u
X1  inAp inAm inBp inBm bias op out    gilbert_core
.ends gilbert_mul
*
.subckt xor_xnor  ia ib bias outa outb
```

```
M1   3   bias   0    0        orbit_nm1_du1 W=12u L=4u
M2   5   5      3    0        orbit_nm1_du1 W=6u L=4u
M3   7   ia     5    0        orbit_nm1_du1 W=6u L=4u
M4   9   ib     7    0        orbit_nm1_du1 W=6u L=4u
M5   5   ia    11    Vdd      orbit_pm1_du1 W=6u L=4u
M6  11   ib     9    Vdd      orbit_pm1_du1 W=6u L=4u
M7   9   9     Vdd   Vdd      orbit_pm1_du1 W=6u L=4u
M8  outa 9     Vdd   Vdd      orbit_pm1_du1 W=12u L=4u
M9  outa bias   0    0        orbit_nm1_du1 W=6u L=4u
M10 13   9     Vdd   Vdd      orbit_pm1_du1 W=12u L=4u
M11 13   13     0    0        orbit_nm1_du1 W=6u L=4u
M12 15   bias   0    0        orbit_nm1_du1 W=6u L=4u
M13 15   15    Vdd   Vdd      orbit_pm1_du1 W=6u L=4u
M14 outb 15    Vdd   Vdd      orbit_pm1_du1 W=6u L=4u
M15 outb 13     0    0        orbit_nm1_du1 W=6u L=4u
.ends xor_xnor
*
.endl multipliers
```

B.3 Sample Hopfield net listing

Below is shown the listing for a Hopfield net where the vector [-1,-1,1,1,-1,-1] is stored. Since this is only meant to simulate the retrieval phase, the XNOR-circuit and memory are not included. Load capacitances were taken from the layout of the chip.

```
file ~/sp/hop1/6na-a.sp: 30 weights - 6 neurons
*
.options ingold=1 brief co=132 nopage probe
 +       dv=0.1 gmin=1e-16 gmindc=1e-16 pivtol=0.4e-16 gramp=3
.global Vdd
*
.tran 1us 500us
.print tran V(out1) V(out2) V(out3) V(out4) V(out5) V(out6)
*
.param nsval=2.98 psval=3.02
$
V0    Vdd      0     5V
V1    ref      0     3.00V
V2    amp_bias 0     0.85V
V4    mulref1  0     2.5V    $ reference for weights
$weightmatrix:
$5_11 w11      0     zero    $          stored vectors:
V5_21 w21      0     2.520   $
V5_31 w31      0     2.480   $
V5_41 w41      0     2.480   $      1   1  -1  -1   1   1
V5_51 w51      0     2.520   $
V5_61 w61      0     2.520   $
$
V5_12 w12      0     2.520   $
$5_22 w22      0     zero    $
V5_32 w32      0     2.480   $         weight  matrix:
```

```
V5_42 w42      0     2.480   $
V5_52 w52      0     2.520   $       0   1  -1  -1   1   1
V5_62 w62      0     2.520   $
$                                    1   0  -1  -1   1   1
V5_13 w13      0     2.480   $
V5_23 w23      0     2.480   $      -1  -1   0   1  -1  -1
$5_33 w33      0     zero    $
V5_43 w43      0     2.520   $      -1  -1   1   0  -1  -1
V5_53 w53      0     2.480   $
V5_63 w63      0     2.480   $       1   1  -1  -1   0   1
$
V5_14 w14      0     2.480   $       1   1  -1  -1   1   0
V5_24 w24      0     2.480   $
V5_34 w34      0     2.520   $
$5_44 w44      0     zero    $
V5_54 w54      0     2.480   $
V5_64 w64      0     2.480   $
$
V5_15 w15      0     2.520   $
V5_25 w25      0     2.520   $
V5_35 w35      0     2.480   $
V5_45 w45      0     2.480   $
$5_55 w55      0     zero    $
V5_65 w65      0     2.520   $
$
V5_16 w16      0     2.520   $
V5_26 w26      0     2.520   $
V5_36 w36      0     2.480   $
V5_46 w46      0     2.480   $
V5_56 w56      0     2.520   $
$5_66 w66      0     zero    $
$
V6    mulref2  0     3.0V    $ ref for neuronal inputs to weights
V7    mul_bias 0     0.77V
V8    lock_en  0     pwl (0s 5V, 250us 5V, 251us 0V, 500us 0V)
V9a   lock_v1  0     nsval
V9b   lock_v2  0     nsval
V9c   lock_v3  0     nsval
V9d   lock_v4  0     nsval
V9e   lock_v5  0     nsval
V9f   lock_v6  0     nsval
*
.lib '../lib/lib1.lib' amplifiers
.lib '../lib/lib1.lib' multipliers
*
.param Cval=1e-12
$
.subckt neuron      act ref bias out lock_val lock_en
  V1   act  inp  0V  $ amperemeter for input current
  $
  $ pass-gate for lock signal:
  M1   out        lock_en    lock_val  0    orbit_nm1_du1 W=6u L=6u
  M2   out        lock_en_i  lock_val  Vdd  orbit_pm1_du1 W=6u L=6u
  $
```

```
  $ inverter for enable-signal for locking:
  M3   lock_en_i  lock_en    0         0    orbit_nm1_du1 W=6u L=6u
  M4   lock_en_i  lock_en    Vdd       Vdd  orbit_pm1_du1 W=18u L=6u
  $
  X1 inp  ref bias  out   i_amp
  Cl out 0  C=Cval
  C2 act 0  C=0.35707e-12
.ends neuron
$
Xn1 act1 ref amp_bias out1 lock_v1 lock_en   neuron Cval=1.24742e-12
Xn2 act2 ref amp_bias out2 lock_v2 lock_en   neuron Cval=1.21995e-12
Xn3 act3 ref amp_bias out3 lock_v3 lock_en   neuron Cval=1.19206e-12
Xn4 act4 ref amp_bias out4 lock_v4 lock_en   neuron Cval=1.17702e-12
Xn5 act5 ref amp_bias out5 lock_v5 lock_en   neuron Cval=1.17560e-12
Xn6 act6 ref amp_bias out6 lock_v6 lock_en   neuron Cval=1.17417e-12
$
Xw12 w12 mulref1 mulref2 out1 mul_bias act2  gilbert_mul
Xw13 w13 mulref1 mulref2 out1 mul_bias act3  gilbert_mul
Xw14 w14 mulref1 mulref2 out1 mul_bias act4  gilbert_mul
Xw15 w15 mulref1 mulref2 out1 mul_bias act5  gilbert_mul
Xw16 w16 mulref1 mulref2 out1 mul_bias act6  gilbert_mul
$
Xw21 w21 mulref1 mulref2 out2 mul_bias act1  gilbert_mul
Xw23 w23 mulref1 mulref2 out2 mul_bias act3  gilbert_mul
Xw24 w24 mulref1 mulref2 out2 mul_bias act4  gilbert_mul
Xw25 w25 mulref1 mulref2 out2 mul_bias act5  gilbert_mul
Xw26 w26 mulref1 mulref2 out2 mul_bias act6  gilbert_mul
$
Xw31 w31 mulref1 mulref2 out3 mul_bias act1  gilbert_mul
Xw32 w32 mulref1 mulref2 out3 mul_bias act2  gilbert_mul
Xw34 w34 mulref1 mulref2 out3 mul_bias act4  gilbert_mul
Xw35 w35 mulref1 mulref2 out3 mul_bias act5  gilbert_mul
Xw36 w36 mulref1 mulref2 out3 mul_bias act6  gilbert_mul
$
Xw41 w41 mulref1 mulref2 out4 mul_bias act1  gilbert_mul
Xw42 w42 mulref1 mulref2 out4 mul_bias act2  gilbert_mul
Xw43 w43 mulref1 mulref2 out4 mul_bias act3  gilbert_mul
Xw45 w45 mulref1 mulref2 out4 mul_bias act5  gilbert_mul
Xw46 w46 mulref1 mulref2 out4 mul_bias act6  gilbert_mul
$
Xw51 w51 mulref1 mulref2 out5 mul_bias act1  gilbert_mul
Xw52 w52 mulref1 mulref2 out5 mul_bias act2  gilbert_mul
Xw53 w53 mulref1 mulref2 out5 mul_bias act3  gilbert_mul
Xw54 w54 mulref1 mulref2 out5 mul_bias act4  gilbert_mul
Xw56 w56 mulref1 mulref2 out5 mul_bias act6  gilbert_mul
$
Xw61 w61 mulref1 mulref2 out6 mul_bias act1  gilbert_mul
Xw62 w62 mulref1 mulref2 out6 mul_bias act2  gilbert_mul
Xw63 w63 mulref1 mulref2 out6 mul_bias act3  gilbert_mul
Xw64 w64 mulref1 mulref2 out6 mul_bias act4  gilbert_mul
Xw65 w65 mulref1 mulref2 out6 mul_bias act5  gilbert_mul
*
.include '../fet-models/fet-param2.mod'
*
```

```
.alter
V9f   lock_v6  0     psval
.alter
V9f   lock_v6  0     nsval
V9e   lock_v5  0     psval
.alter
V9f   lock_v6  0     psval
V9e   lock_v5  0     psval
.alter
V9f   lock_v6  0     nsval
V9e   lock_v5  0     nsval
V9d   lock_v4  0     psval

         .
         .
         .  64 times
         .
         .
.alter
V9f   lock_v6  0     psval
V9e   lock_v5  0     psval
V9d   lock_v4  0     psval
V9c   lock_v3  0     psval
V9b   lock_v2  0     psval
v9a   lock_v1  0     psval
.end
```

Appendix C

Miscellaneous

This appendix contain some material that did not fit in to the main part of the thesis: UV-conductor models that may be interesting for other applications of UV-programming, and an attempt to confirm them experimentally. A parasitic capacitance measurement with a strange result. The result of a UV-lamp that did not emit short enough wavelenghts.

C.1 UV-conductor models

C.1.1 Linear model

As I stated in the chapter about the UV-reference, the charging of the floating node in a reference circuit may be described by the equation

$$\frac{V_{dd} - V_{fn}}{R_1} = \frac{V_{fn}}{R_2} + C\frac{dV_{fn}}{dt} \tag{C.1}$$

which was arrived at by looking at figure 6.4. R_1 and R_2 denotes the resistance of the UV-conductors, C is the total floating-node capacitance, and V_{fn} is the floating-node voltage. (C.1) is a first order linear differential equation with the solution

$$V_{fn} = \left(V_0 - \frac{b}{a}\right) e^{-at} + \frac{b}{a} \tag{C.2}$$

straight from the textbook. V_0 is the starting voltage of the floating-node, $a = \frac{R_2+R_1}{R_1R_2C}$, and $b = V_{dd}$ when $t \geq 0$. However, this is valid only for linear UV-conductors.

C.1.2 Maher's model

According to Maher [37], the UV-conductors have a resistance described by the equation

$$R = \beta\,(V_R)^{-\frac{1}{2}} \tag{C.3}$$

where V_R is the voltage across the UV-conductor. The constant β incorporates parameters silicon dioxide thickness, exposed edge length, and UV-light intensity/efficiency. By replacing R_1 and R_2 in equation (C.1) by (C.3), we get

$$\frac{(V_{dd} - V_{fn})^{\frac{3}{2}}}{\beta_1} = \frac{(V_{fn})^{\frac{3}{2}}}{\beta_2} + C\frac{dV_{fn}}{dt} \tag{C.4}$$

This is a first order non-linear differential equation which I am not capable of solving analytically.

C.1.3 Benson and Kerns' model

The same holds for the function adopted by Benson and Kerns [7]:

$$I_{UV} = I_{leak} + GV_R - V_0\,(G-g)\tanh\left(\frac{V_R}{V_0}\right) \tag{C.5}$$

where I_{leak} is a parasitic leakage current, V_R is the voltage across the UV-conductor, and G and g are constants. Replacing the terms in equation (C.1) with (C.5) gives

$$G(V_{dd}-V_{fn}) - V_0(G-g)\tanh\left(\frac{V_{dd}-V_{fn}}{V_0}\right) = GV_{fn} - V_0(G-g)\tanh\left(\frac{V_{fn}}{V_0}\right) + C\frac{dV_{fn}}{dt} \tag{C.6}$$

As this model has come to my attention rather late, I have not tried to solve this equation analytically, but I suspect that it cannot be done.

C.1.4 Model comparisons

Fortunately, it is quite easy to put these models into hSPICE. Doing this has the added advantage that effects in a sensing follower may be modelled.

If the charging time constant is defined to be the time it takes for the floating-node to charge from starting point to 63% of the final voltage, it is easy to see that if the UV-conductors are non-linear, the time constants will change with changing power supply.

Using hSPICE, we can see that this is correct. Figure C.1 shows a floating-node with linear conductors, versus a floating-node with conductors that behave according to Maher's model. An ad hoc "normalization" has been done for $V_{dd} = 5$ volts.

In figure C.2, a comparison between the time constants of figure C.1 and measured time constants has been done. We see that the curve of measured time constants has a minimum. Because the measured floating-node charging curves were put at a starting point by influence of a coupling capacitor, the range of curves for a given starting point was limited by capacitive division. For some reason, moving the starting point to measure a different set of curves also moved the minimum time constant. Clearly something strange was going on, which I am unable to explain. The measurements do not fit either of the proposed models.

C.2 Influence of C_{gs} parasitic

After programming of a floating-node, the stored charge will be influenced by parasitic capacitors of the sensing transistor. In an attempt to demonstrate the parasitics, a floating-node connected to one transistor of a differential pair was charged to 1.2 Volts. During programming the input, the gate of the other transistor of the differential pair, was held at 2.5 Volts. The bias voltage was 0.8 volt, just below threshold. The drains of the two transistors were each connected to p-transistors in diode configurations. After programming, the input was swept through the range 0.8–4 volts. During the sweep, the floating-node voltage was monitored.

The intention was to measure the influence of the parasitic gate-source capacitor C_{gs} on the floating-node voltage. When the input exceeded the floating-node voltage, the common source of the differential pair followed the input, with a gain less than one because of body-effect. Therefore a stable floating-node voltage was expected, up to the point where the input equalled the floating-node voltage. Then floating-node voltage would increase because of the parasitic capacitor. A faint curvature of the graph was expected because of change in body-effect with increasing source-bulk voltage.

The measure curve is shown in figure C.3. It is close to the expected, except for the very curious "dip" around 1.2 volts. The experiment was repeated for a different floating-node voltage, and the dip again occured at the stored voltage. The dip must be caused by an decrease in capacitance, giving off electrons to other capacitors, thereby lowering the measured voltage.

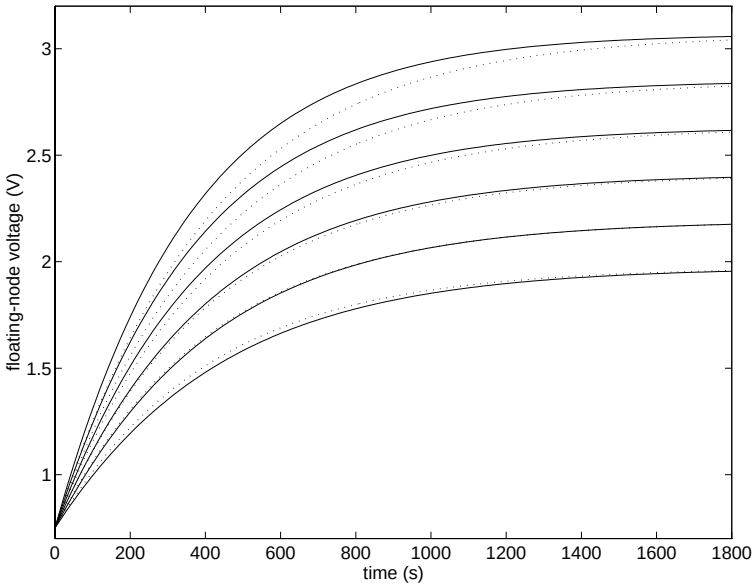


Figure C.1: *Maher's model versus a linear model, simulated in hSPICE. Dotted lines are the linear model, solid lines are Maher's model.*

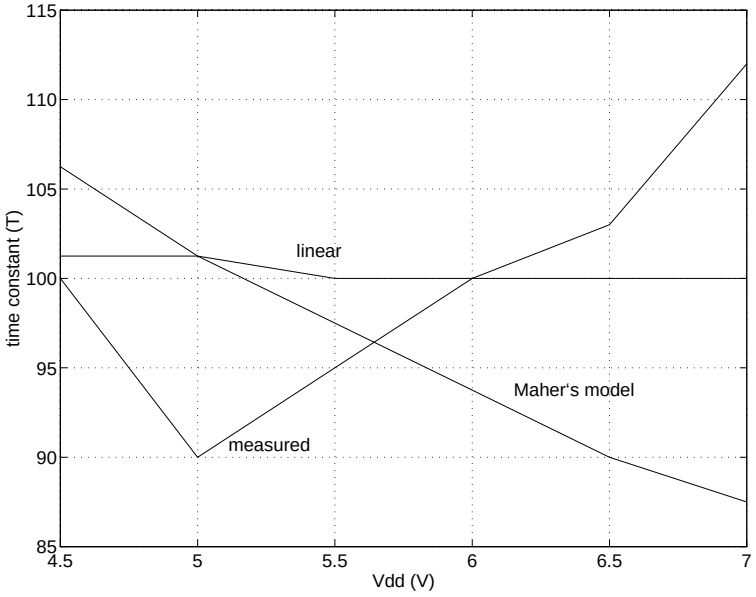


Figure C.2: *Measured time constants versus simulated.*

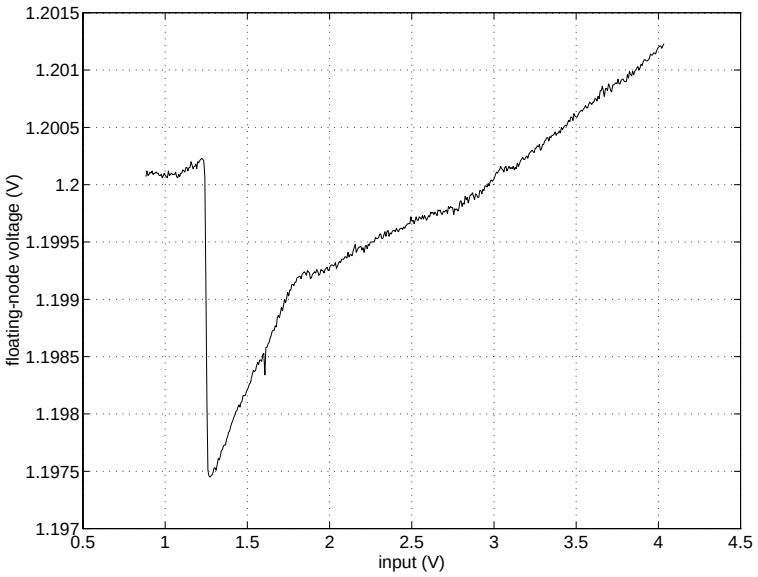


Figure C.3: *Measured influence of the sensing transistor on stored voltage. The measurement was done on a differential pair as shown.*

According to textbooks (e.g. [53]), gate capacitance decrease with increasing gate-source voltage, but only in moderate inversion, and only gradually. The measured dip is very abrupt.

The stored voltage was sensed by a follower, which could influence the measurements itself. However, conditions at the follower input should be reasonably invariant, because both its input transistors always see the same voltage (excepting offset). Also, the change in voltage on the floating-node was miniscule, because of capacitive division. The sensing follower should not influence observation significantly.

The total voltage swing is small, only 4 millivolts. But the total floating-node capacitance is here rather large, estimated at 1.14 pF. For smaller capacitances, which are desireable to decrease programming time, the effect should increase.

The voltmeter used was a Keithley 617.

C.3 The SpotCure 400 Watt UV-lamp

We had at our disposal two UV-light sources, both purchased from the same supplier. One was a standard mercury EPROM eraser with an output of 4 Watts. This has already been described, as it was used for all the measurements presented earlier.

The other UV-light source at our disposal was a UVP SpotCure with a total output of 400 Watts. The wavelength was specified by the supplier to have the range 250-450 nm. It also outputted a substantial amount of visible (blue) light. Effect of this light on circuits was not negligible, so an optical bandpass filter was placed on top of the chip under test. This filter had a passband from 330 to 370 nm with 3dB attenuation. From the source, a fibre optic cable guided the light to the chip under test. The end of the fibre optic cable could be regulated to different distances from the chip. Light could be admitted into the cable by a mechanical shutter, operated by a computer, and by a built in timer. The chip under test was exposed to UV-light for a certain time, then the timer turned the light off. A computer then made a measurement. When the sample was completed, the computer turned the UV-light back on, resetting the timer.

In late fall 1992 I started measurements on the first batch of chips. Using the SpotCure, the end of the fibre optic cable was placed at a distance of seven cm from the chip under test.

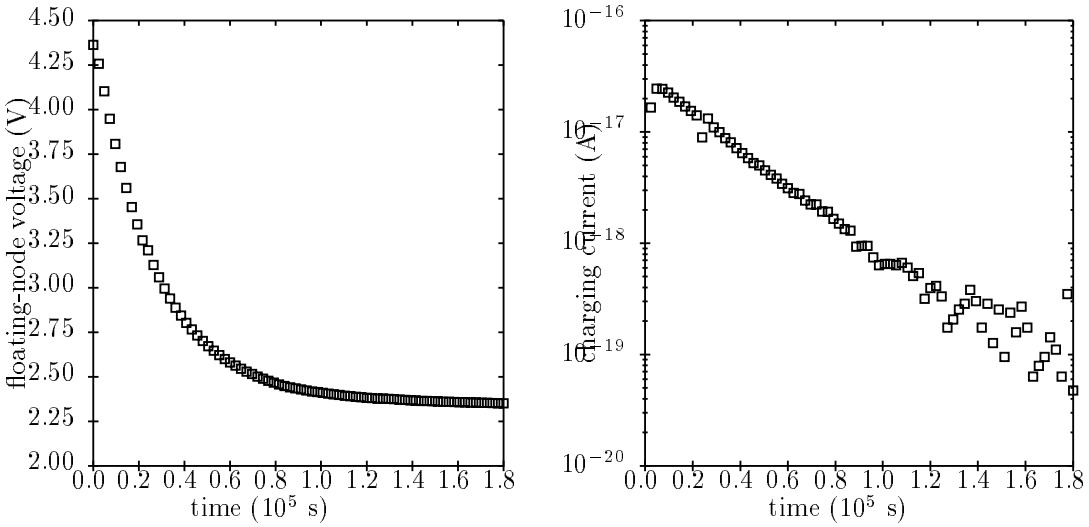


Figure C.4: *Left: Measured charging curve of a type 'A' voltage reference using the SpotCure UV-lamp. Right: Current into the floating-node, computed from the data in the figure to the left. The floating-node capacitance was 380 fF.*

Supply voltage was 5 Volts. The chip was exposed to UV-light for 40 minutes, then a sample was taken, before a new exposure period started. The measured charging curve is shown to the left in figure C.4. The reference obviously came from processing with a stored charge, so figure shows a discharge curve rather than a charging curve. The time constant is approximately 2500 seconds. This is the only successful measurement using this light source. The extremely long programming time gave ample opportunity for interruptions, either by curious fellow students, the computer department, or failing chips (three times).

The capacitance of the floating node was used to compute the charging current, shown to the right in figureC.4. We see that the current start in the 10^{-17} range. This was many orders lower than I had been led to expect beforehand. From Williams [64] I later found that the "long" wavelength light admitted by the band pass filter would reduce programming current, the light not being able to give enough energy to electrons in the silicon. Findings by Kerns et. al. [27] support this. I tried to remove the filter, we got a replacement UV-light tube from the supplier (emitting light with shorter wavelengths), I tried a different fibre optic cable and no fibre optic cable at all. None of this gave any significant improvements. Perhaps needless to say, this lamp was a disappointment, wasting several months of my time. Fortunately, the power supply fuse burned out, and while searching for a replacement fuse I tried the old EPROM eraser, which turned out to work much better, and was used for all the subsequent programming.

Finally, just to satisfy my curiosity, I tore out all the innards of the SpotCure except the light emitting tube, and placed a chip as close to the tube as possible. This gave considerable shorter charging time than the EPROM eraser. It also gave considerable larger offsets in the circuits on the chip (several hundred milliVolts), besides giving me second degree burns on three fingertips when I touched the chip afterwards.

Appendix D

Papers

The work for this thesis resulted in two papers, both co-authored by T.S.Lande and me. They are included on the following pages.

Abusland A., Lande T. S:
Local Generation and Storage of Reference Voltages In CMOS Technology using UV-Light.
Proc. 11th European Conf. Circuit Theory and Design, Vol. I, 1993, pp. 281–286.

Abusland A., Lande T. S:
An Analog Continuous-Time Micropower Hopfield Net.
To be presented at the IEEE Internat'l Conf. Neural Networks, June 26th–July 2nd, 1994.